The Kinetics and Mechanisms of the Gas-Phase Pyrolysis of Environmentally Significant Organochlorine Compounds.

Thesis submitted for the degree of

Master of Philosophy

at the University of Leicester

by

Vinton James Carter BSc (Bangor)

School of Environmental Sciences

Nene College of Higher Education

July 1997

Contents

Acknowledgements

I would like to acknowledge the aid and support of the following people, particularly Dr Mark Pennington whose untiring assistance has been invaluable :

Dr Mark Pennington	(1st Supervisor)
Dr David Symon	(2nd Supervisor)
Prof Malcolm Fox	(External Supervisor)

In addition, I owe a debt gratitude to everyone else at Nene College who has helped me in completing this work, specifically the technical staff of the School of Environmental Science and the staff of the Nene Centre for Research.

Also, I wish to thank Nene College of Higher Education for financially supporting this project.

Glossary of Abbreviations

Abbreviation	*Meaning*
2,3,7,8-TCDD	2,3,7,8-tetrachlorodibenzodioxin
2,4,5-T	2,4,5-trichlorophenoxyethanoic acid
4,4'-DDM	1-chloro-2,2-bis(4-chlorophenyl)ethene
CFC	chlorofluorocarbon
DDE	1,1-dichloro-2,2-bis(4-chlorophenyl)ethene
DDT	1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane
DTA	differential thermal analysis
DTG	differential gravimetric analysis
GC	gas chromatography
GSV	gas sampling valve
LC_{50}	concentration of the pesticide in the medium in which the test species lives at which 50% of the sample population dies.
LD_{50}	dose at which 50% of the sample population dies.
lg	$\log_{10}$
MS	mass spectrometry
MSWI	municipal solid waste incineration
PCDD	polychlorinated dibenzo-*p*-dioxin
PCDF	polychlorinated dibenzo-*p*-furan
PIC	product of incomplete combustion
ppb	parts per billion
PTFE	polytetrafluoroethene
RCEP	Royal Commission on Environmental Pollution
TA	thermal analysis
TCE	1,1,1-trichloroethane

Aims

To investigate the mechanisms of the gas-phase pyrolyses of organochlorine compounds of environmental interest, more particularly those that might have a bearing on or be involved in incineration processes, either through inclusion in domestic waste or as components of toxic waste materials.

To develop the technique of pulse stirred-flow to the area of organochlorine pyrolyses as an inexpensive, relatively simple and effective method of mechanistic investigation.

Chapter 1 : Introduction

Environmental Impact of Organochlorine Compounds

Uses and Importance of organochlorine compounds as industrial materials

Organochlorine compounds remain one of the most widely used groups of chemicals in the World. The applications in which they are common include high strength polymeric materials, e.g. polyvinyl chloride (PVC); a variety of common solvents, e.g. dichloromethane; aerosol propellants, e.g. chlorofluorocarbons and many other gases with a diverse range of applications,[1] e.g. the widely used anaesthetic 1,1,1-trifluoro-2-bromo-2-chloroethane (halothane).

One of the reasons that organochlorine compounds were useful in such a wide range of applications is their generally high thermal and electrical stability. Chlorofluorocarbons (CFCs) and polychlorinated biphenyls (PCBs) are prime examples of classes of compounds with high thermal stability. CFCs have been used in refrigeration systems where they undergo many cooling and heating cycles, yet have long lifetimes. PCBs (Figure 1) were used extensively as electrical and thermal insulators, applications in which their stability is essential. Unfortunately, it is this very stability that has contributed to the detrimental effect of organochlorine compounds on the environment. As an enormous number of commonly used organochlorine compounds have some toxicity to at least one group or species, the fact that they have long lifetimes in the environment is a major factor for consideration.

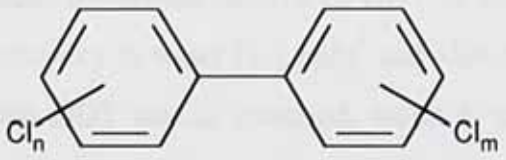


Figure 1 General structure of Polychlorinated Biphenyls (PCBs).

n and m may take values 1-5

Pollution by Organochlorine Compounds

Pesticides

1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane (DDT) provides an excellent example of the environmental effects that can result from organochlorine pesticide use. Furthermore, this compound has been well studied[2-9] in many of its environmental effects and there is, therefore, much in the literature upon which to draw.

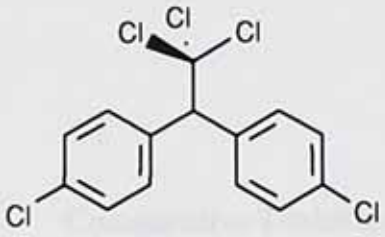


Figure 2 1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane (DDT)

Although DDT was synthesised first by Zeidler (1874), it wasn't recognised as a useful insecticide until many years later by Paul Müller (1939).[2a] In the initial period of its use DDT was felt to embody many of the virtues necessary in a good insecticide: it had a very broad spectrum of insecticidal activity (i.e. it was effective against a large number of species of insects), it has relatively low mammalian toxicity under conditions of normal use (Table 1) and it was both simple and cheap to manufacture. Due to these virtues, DDT was applied throughout the World in enormous quantities (~ 1 x 10^5 tonnes / annum) in the 1950s and 1960s.[2b] However, further studies found that DDT remains in the environment for long periods of time after application (estimated lifetime of DDT in soil is 5-11 years).[3,4] This is due, partly, to its very low solubility in water (1.2 ppb)[5] and also to its strong adsorption onto certain soil particles.[6] Of the DDT that is removed, much is through conversion to other compounds which have virtually the same impact on the environment, e.g. 1,1-dichloro-2,2-bis(4-chlorophenyl)ethene (DDE) which, although non-insecticidal, does have detrimental effects for many varieties of fish and birds, e.g. suppression of fertility,[7] and 1,1-dichloro-2,2-

bis(4-chlorophenyl)dichloroethane (DDD) which is actually more toxic towards insects than DDT.

As might be expected from the low aqueous solubility exhibited by DDT, it has a high lipid solubility. This leads to bioaccumulation, a phenomenon quite common in the environmental chemistry of organochlorine compounds, and results in concentrations in higher organisms up to four orders of magnitude higher than in the environment.[2c] In the case of DDT, bioaccumulation can lead to sterility, weakening of the shells of birds' eggs and outright death.[7] The most vulnerable species are those at the top of the food chain, usually predators, e.g. falcons and eagles, because it is in these that the greatest concentration increase due to bioaccumulation occurs.

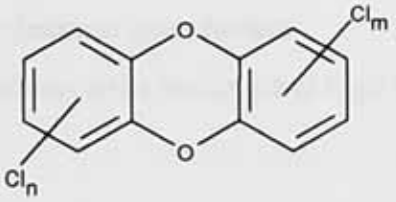
Table 1 Comparative Toxicities for DDT

Compound	LD_{50} by contact mg / kg		LC_{50} $\mu g\ dm^{-3}$
	Insects[8]	Mammal[8]	Shellfish[9]
DDT	10-30	3000	0.6 - 60

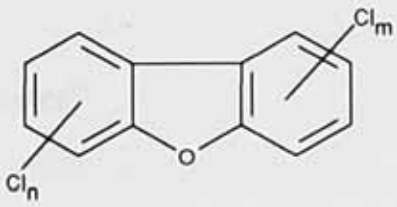
Polychlorinated dibenzodioxins / furans

Structure and Toxicity

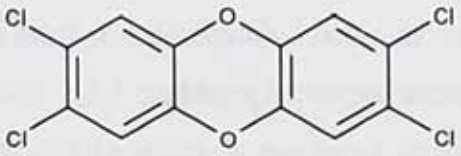
Polychlorinated dibenzo-*p*-dioxins (PCDDs) and polychlorinated dibenzo-*p*-furans (PCDFs) have received much attention due to their high toxicities and other undesirable effects. There are several formulae which fit each of the groups PCDDs and PCDFs and each formula has a number of possible isomers (Figure 3) due to the positioning of the chlorine atoms (PCDFs exhibit more isomers due to their greater relative asymmetry). It has been found that the tetra-chlorinated members of the PCDDs have the greatest toxicity. Indeed, one such compound, 2,3,7,8-tetrachlorodibenzodioxin (2,3,7,8-TCDD), is one of the most toxic compounds known to man with LD_{50} (oral) to rats ≈ 30 μg / kg[2d] (although this figure varies greatly depending on the species).



General Structure of polychlorinated dibenzo-*p*-dioxins
m+n >=2 and 1 < m,n < 5



General Structure of polychlorinated dibenzo-*p*-furans
m+n >=2 and 1 < m,n < 5



2,3,7,8-tetrachlorodibenzo-*p*-dioxin (2,3,7,8-TCDD)

Figure 3 Showing General structures for PCDDs and PCDFs and the PCDD 2,3,7,8,-TCDD

Teratogenicity, carcinogenicity and mutagenicity

PCDDs / PCDFs have long been known to cause teratogenic effects (birth defects) in many species,[10] yet the perception that these compounds are directly carcinogenic or mutagenic is debatable. Studies have been attempted to ascertain the extent to which 2,3,7,8-TCDD is a carcinogen but they have had marginal or conflicting results.[11] Indeed, Smith *et al*[12] stated

that all but one study performed had used insufficient numbers of subjects having had significant exposure to the compound to provide any evidence for either side of the argument. While it is certain that PCDDs are extremely harmful and undesirable materials, environmentally, it is possible that their risk has been exaggerated. The most serious incident to date involving PCDDs was in Italy, at Seveso, in 1976 when approximately 3 kg of 2,3,7,8-TCDD, together with other substances, were released into the surrounding countryside.[13] Despite several injuries and the trauma of evacuation there were no fatalities resulting directly from this incident, the only effects apparent being outbreaks of chloracne and some reports of spontaneous abortion.

Sources of PCDDs and PCDFs in the Environment

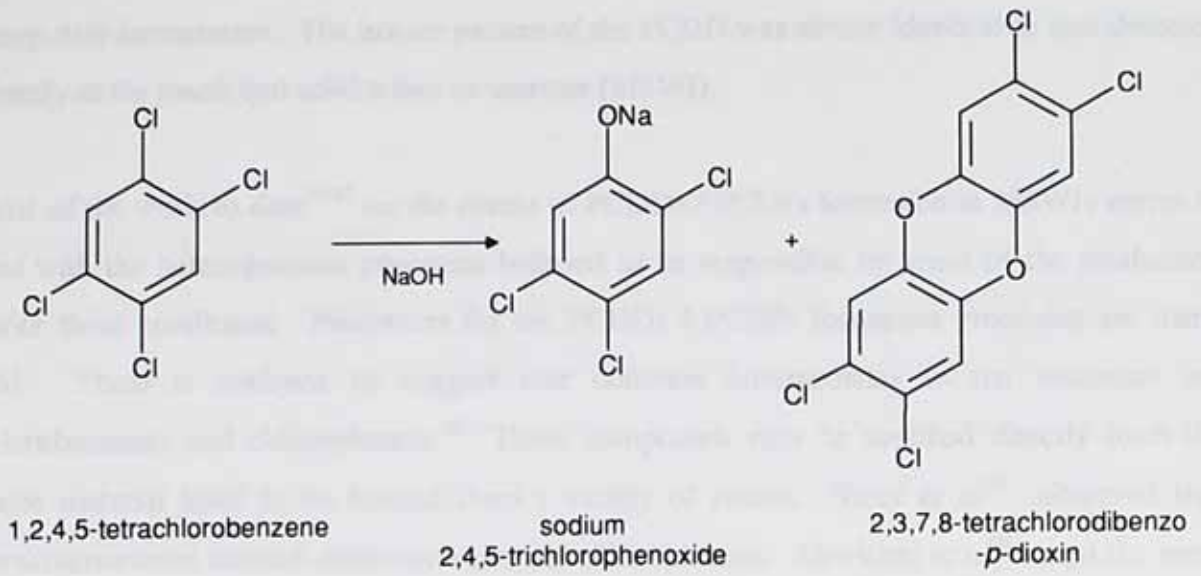
Sources of PCDDs / PCDFs in the environment can be separated into two types

I. Direct chemical manufacture

II. By-products from burning and high temperature processes[14]

Direct chemical manufacture

Some syntheses, especially those involving aromatic compounds, can result in the production of PCDDs. For example, the herbicide commonly known as Agent Orange, used extensively in the Vietnam war, was actually 2,4,5-trichlorophenoxyethanoic acid (2,4,5-T) contaminated with large amounts of 2,3,7,8-TCDD which is produced during the manufacturing process (Scheme 1).



Scheme 1 First step in the synthesis of 2,4,5-T showing 2,3,7,8-TCDD production

By-products from burning and high temperature processes

Heterogeneous Processes

There is evidence for the production of PCDDs / PCDFs from a number of burning processes involving chlorinated starting materials. For example, it has been shown that soil, on which electrical equipment has been burned for metal reclamation, contains substantial amounts of PCDDs.[15] PCDDs / PCDFs have been detected in transformer fires involving oils and insulation materials containing polychlorinated biphenyls (PCBs),[16] in wood ash from wood burning stoves,[17] and from rice straw smoke (at levels three orders of magnitude higher than in the surrounding environment).[18]

However, the present work is concerned mainly with the formation of PCDDs / PCDFs in incinerators. Some municipal waste incinerators are known to release PCDDs / PCDFs into the environment in significant amounts. One example of this is a study by Kocan *et al*[19] that found relatively high levels of PCDDs in the hair of people scavenging on a municipal waste

dump near incinerators. The isomer pattern of the PCDD was almost identical to that detected directly at the municipal solid waste incinerator (MSWI).

Most of the work to date[21-27] on the routes of PCDDs / PCDFs formation in MSWIs seems to deal with the heterogeneous processes believed to be responsible for most of the production under these conditions. Precursors for the PCDDs / PCDFs formation processes are many fold. There is evidence to suggest that common intermediates in the syntheses are chlorobenzenes and chlorophenols.[20] These compounds may be supplied directly from the waste material itself or be formed from a variety of routes. Tirey *et al*[21] observed that tetrachloroethene formed chlorobenzenes and chlorophenols. Altwicker *et al*[22] noted the same type of formation from other small chlorinated compounds. Other sources of these aromatic intermediates include various polymers. For example, large amounts of simple chlorinated aromatics were observed, associated with soot particles, from the pyrolysis of polyvinyl chloride and polyvinylidene dichloride.[23]

The heterogeneous reactions forming both the precursors and the dioxins themselves are concentrated on two types of surfaces. Fly-ash, containing metals in various oxidation states, has shown a high catalytic activity for these reactions and the particles of ash often have much greater concentrations of chlorinated aromatics, including PCDDs / PCDFs, than the flue gases. Of several metal compounds tested, those containing ionic copper have shown the highest activity.[24] The ash often contains other components such as magnesium aluminosilicates with particulate carbon present. Carbon particles constitute the other type of catalytic surface. Carbon in a variety of particle sizes and textures has been found to be very effective in promoting the *de novo* synthesis of PCDDs / PCDFs. These include a pitch-like material formed as an intermediate in coke and graphite production,[25] and a particulate form of organic carbon.[24]

As the heterogeneous processes seem to be the primary sources of PCDDs / PCDFs in MSWIs it would be advantageous to inhibit these reactions. Several organic / inorganic bases have been found to be very effective in doing this with up to 99% inhibition of PCDD's / PCDF's production being achieved.[26] Also it is worth noting that not all chlorinated compounds lead to PCDD's / PCDF's formation. A study of chlorofluorocarbon (CFC) destruction in MSWIs

showed greater than 99.9% destruction without production of PCDDs / PCDFs or exceeding the HF and HCl emission levels.[27]

Homogeneous Processes

There is a dearth of information on this subject, perhaps because heterogeneous processes are perceived to be more important in the production of toxic materials by incineration processes. Primarily, it is to help fill this gap that this study is intended. Although there is little in the literature on homogeneous toxic by-product formation in MSWIs, the field of chlorinated hydrocarbon pyrolysis has been more widely investigated.

Waste Disposal

Modern municipal solid waste incineration is a two stage process that is designed to transform the overall oxidation state of the materials present in the waste, thus rendering them safe for disposal. Reactors vary in design but one of the most common is the rotary kiln which consists of an inclined rotating tube into the elevated end of which the waste is added. The waste moves down the heated tube emerging from the lower end as ash, having liberated many decomposition gases such as carbon dioxide, hydrogen chloride, water and volatile organic compounds.

Incineration has the advantages of greatly reducing the volume and mass of the waste, reducing the toxicity of many of the constituents of the waste and, occasionally, of generating sufficient heat to be useful in contributing to the energy requirements of the local community. Unfortunately, the disadvantages of the process are also considerable. Besides being a relatively expensive method of disposing of waste, there is a risk of environmentally damaging gases, e.g. the 'acid rain' gases sulfur dioxide and nitrogen oxides, escaping in the incinerator effluent. The public's perception of municipal solid waste incineration has become increasingly negative due to the possibility of significant release of toxic compounds from incinerators which, frequently, are sited near population centres. Perhaps the most emotive subject of concern is the formation of products of incomplete combustion (PICs) such as the polychlorinated dibenzo-*p*-dioxins (see the previous section).

Landfill, the major alternative to municipal solid waste incineration, refers to the practice of disposing of waste by dumping it in a specially designated and prepared site. Currently, although this is the dominant form of waste disposal in the UK (93% of waste is disposed of in this way), it is likely that it is set to decrease. Whilst having a more benign aspect than waste incineration in the eyes of the public, landfill is gradually losing favour with pollution review / control bodies, such as the Royal Commission on Environmental Pollution (RCEP), who recommended that there be an extra levy charged on it,[28] now established as £7 / tonne. In addition, sites possessing suitable geology to support landfill are becoming scarce and there have been several well publicised leakages of methane gas, sometimes involving explosions,[a] which have devalued numerous properties in the vicinity of existing sites.

As a result, many Waste Disposal Authorities are looking for suitable alternatives to landfill, especially as the very low cost is threatened. In addition to the RCEP's recommended levy, it is possible that the true cost of landfill is much greater than the generally accepted value (£5-6 m^{-3}).[29] Estimates of the true value,[30] through the use of a more realistic gate fee and the cost of treating leachate as determined by the Mogden formula,[b] have been made giving figures as high as £15 m^{-3}. Taking this into account, it seems likely that municipal solid waste incineration will become more important in the future. If this is true, the necessity to address the environmental concerns about MSWI, through fundamental chemical research in the area, cannot be doubted.

---

[a] Loscoe, Derbyshire, late 1980s.

[b] **Mogden Formula**

[from reference in, *Waste Incineration: BPEO?* M. Pennington, D. Symon and V. Carter, 1995, unpublished material.]

Total Cost of treating leachate /pence m^{-3} = 6.02 +7.4 + 11.67 x COD/445 + 14.86 x SS/336

where Chemical Oxygen Demand (COD) is the actual COD and SS is the settleable solids content of the waste to be discharged.

Gas-Phase Pyrolysis of Relevant Model Organochlorine Compounds

Introduction

Research into the pyrolyses of organochlorine compounds began in 1929 when Glass and Hinshelwood[31] reported the pyrolysis of iodopropane in the temperature range 573-623 K and proposed the reaction scheme below (Scheme 2) to account for their results.

$$2\,C_3H_7I \longrightarrow C_3H_6 + C_3H_8 + I_2$$

Scheme 2 Pyrolysis of iodopropane

Since this early work, the field has expanded enormously[32] into one of the most important areas of gas-phase chemistry. A wide range of techniques have been developed to help in the investigation of such reactions. Of these methods, stirred-flow, the basis of the technique used in this work, is one of the most widely applicable and is discussed in Chapter 2.

Rationale for Compound of Study

DDT

DDT, although no longer used legally in many of the industrially developed countries, is still of major importance in the developing and Eastern Block countries. Annual production at the present time (although hard to estimate due to the difficulty of obtaining accurate figures from the manufacturers) is at least 9×10^3 tonnes, a tenth of the peak production which occurred in the late 1950s.[2] Therefore, DDT represents a significant danger to the environment in those parts of the World in which it is used and the problem of its disposal is current. The value of studies aimed at improving our knowledge of possible disposal methods for this compound is obvious.

Also, as fundamental research, the pyrolysis of DDT represents considerable opportunity. The problem of PCDD formation detailed earlier may be a phenomenon observed during DDT's thermal decomposition, especially as there are already certain structural similarities between

DDT and PCDDs, for example, both have chlorine substituted aromatic rings. To date, pyrolysis of DDT has received very little attention in the literature. What material does exist provides solely qualitative data (see below). Such reports are useful in helping to point the direction for subsequent investigation but in themselves represent only a basic outline of the problem.

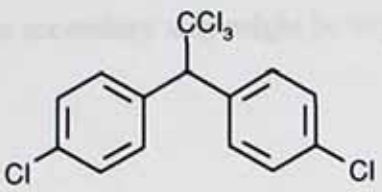
Pyrolysis of 1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane (DDT)

General Thermochemistry

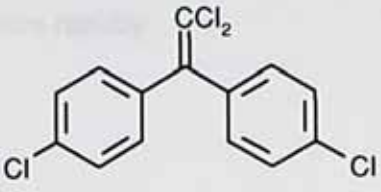
The thermochemistry of a range of chlorinated pesticides has been studied recently[33] using differential thermal analysis (DTA), differential gravimetric analysis (DTG) and thermal analysis (TA). DDT (Figure 4a) started to decompose at ~673 K with a simultaneous loss of mass equivalent to one molecule of hydrogen chloride per molecule of DDT. This, together with the fact that l,l-dichloro-2,2-bis(4-chlorophenyl)ethene (DDE) (Figure 4b) was detected in the sample vessel after heating DDT, led the authors to conclude that dehydrochlorination was the primary process of its thermal behaviour. On comparison of the thermal analysis traces for DDE with those for DDT, it was noticed that the decompositions after the point at which the first mass loss of DDT occurred were very similar, although the DDT trace indicated the presence of some extra side reactions at high temperatures that were not part of DDE's decomposition. These side reactions could be due either to the presence of residual DDT, or to products of the dehydrochlorination (i.e. HCl) retained in the matrix.

Pyrolysis (in a Nitrogen Atmosphere and in Burning Tobacco)

Chopra *et al*[34-37] published a series of papers on the subject of the pyrolysis of DDT and related pesticides in tobacco/cigarette smoke. Part of this study[34,35] was an investigation of the thermal decomposition of DDT in a nitrogen atmosphere at 1173 K. This part of their investigation is directly relevant to this project. The studies of the processes taking place in burning tobacco are less applicable to this study but represent the only work that concentrates on the more complicated reactions that might take place during combustion. Thus some information on DDT's combustion in incinerators may be forthcoming.



(a) 1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane (DDT)



(b) 1,1-dichloro-2,2-bis(4-chlorophenyl)ethene

Figure 4 DDT and one of its derivatives

The products of the pyrolysis in nitrogen, at 1173 K, were found to be[36] (Scheme 3 to Scheme 6):

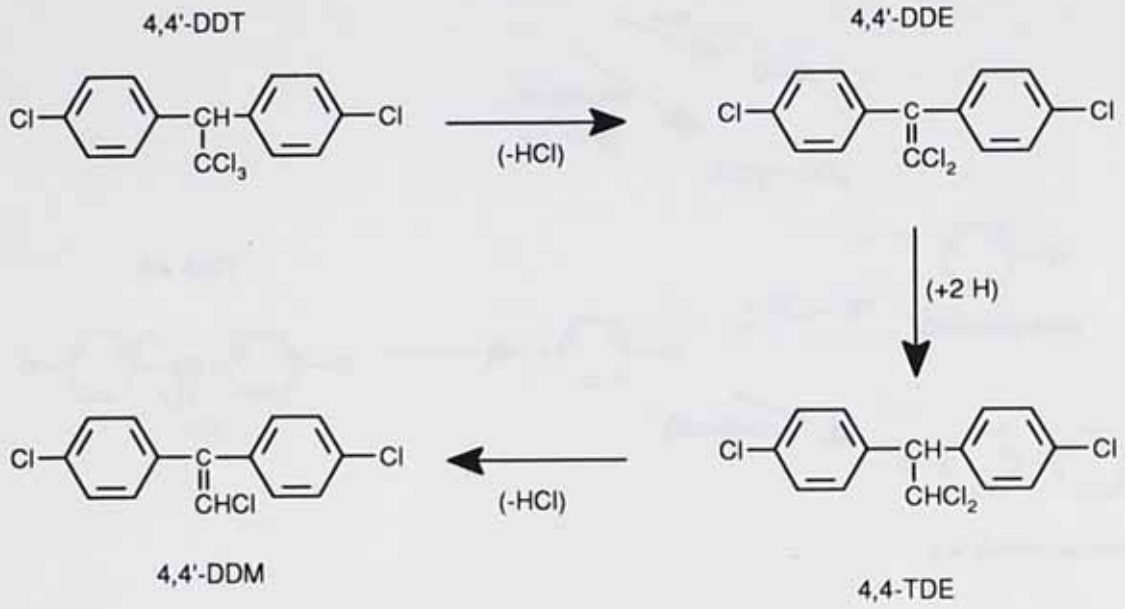
1,1-dichloro-2,2-bis(4-chlorophenyl)ethene (4,4'-DDE)	hexachloroethane
1-chloro-2,2-bis(4-chlorophenyl)ethene (4,4'-DDM)	cis- and trans- dichlorostilbenes
chloro-bis(4-chlorophenyl)methane	bis(4-chlorophenyl)methane
4-chlorobenzyl chloride	4,4'-dichlorobiphenyl
1,1-dichloro-2,2-bis(4-chlorophenyl)ethane (4,4'-DDD)	tetrachloroethene
trichloroethene	tetrachloromethane
trichloromethane	dichloromethane

Chopra *et al*[37] postulated mechanisms for the formation of these products. The mechanisms were grouped into three categories, outlined below.

Mechanisms involving dehydrochlorination and hydrogenation reactions (Scheme 3)

The dehydrochlorination step, similar in nature to that of 1,1,1-trichloroethane, is one of the most facile reactions for DDT. Indeed, Lubkowski *et al*[33] noted it as one of the major processes that takes place on heating the pure material. Therefore its inclusion as the primary and tertiary steps of the mechanism below (Scheme 3) is justified. As the authors[37] point out, however, the secondary step is unlikely to be very fast in an inert atmosphere as the only source of H atoms is DDT itself. The study in question was aimed at elucidating the

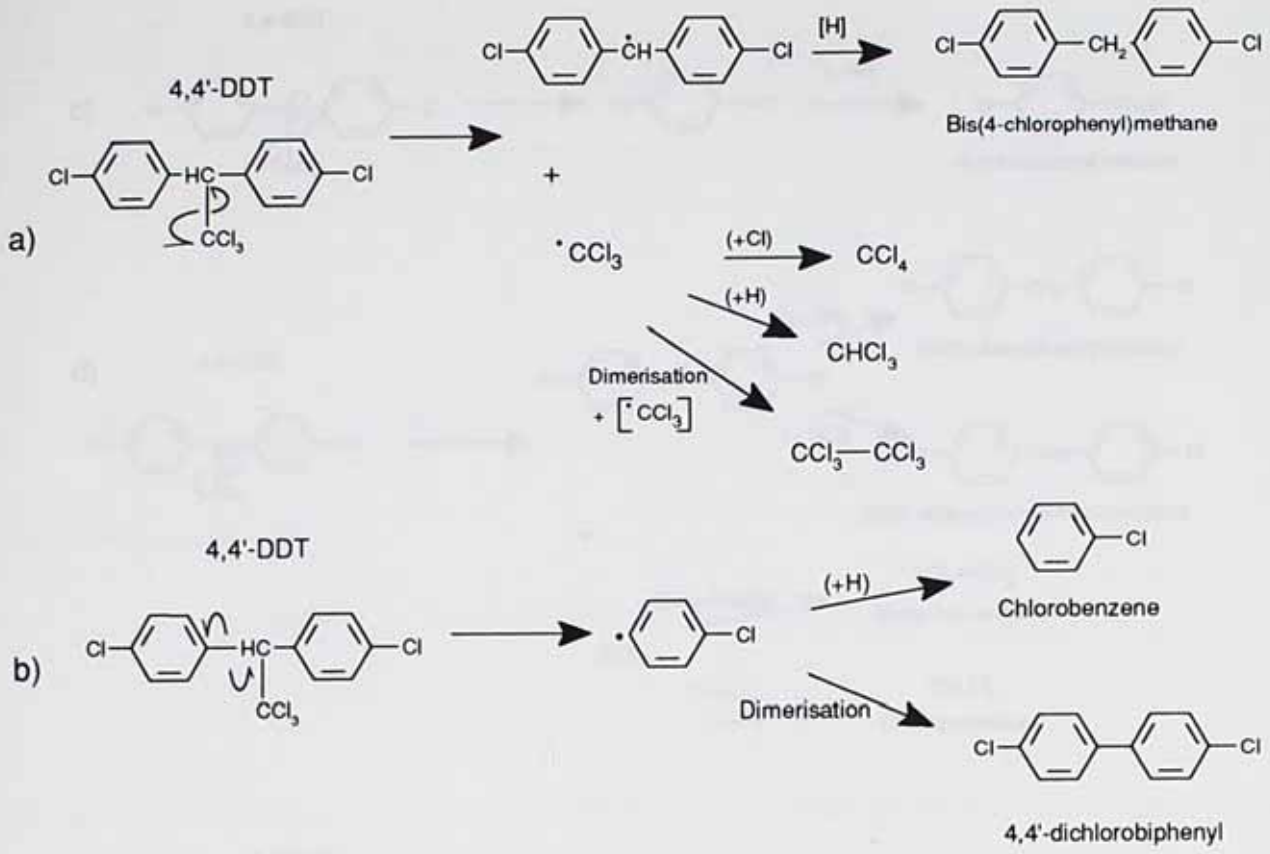
mechanisms of DDT pyrolysis in cigarettes, a highly reducing atmosphere, and, hence, one in which the secondary step might be expected to proceed more rapidly.



Scheme 3 Mechanisms involving dehydrochlorination and hydrogenation reactions

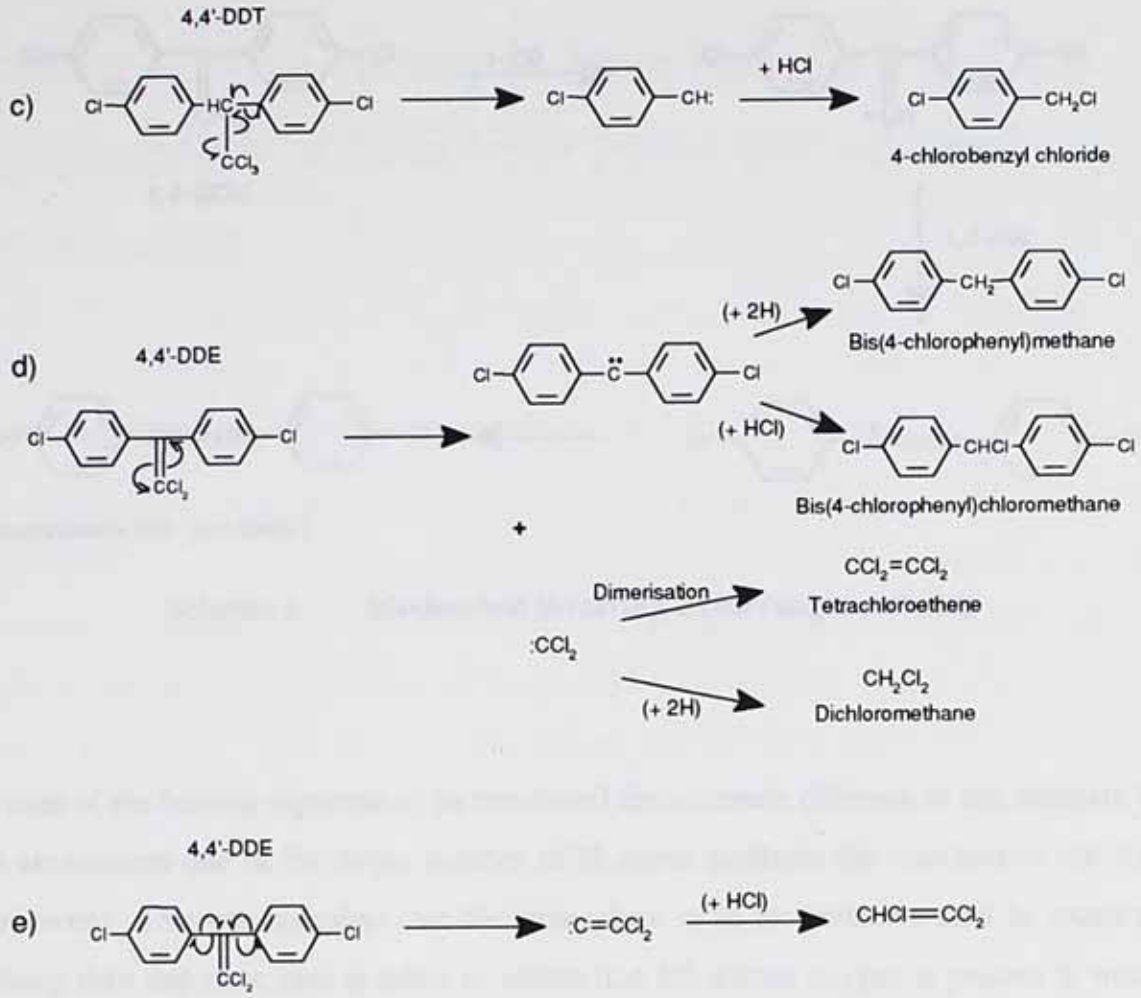
Mechanisms involving the fragmentation of DDT molecule (Scheme 4 and Scheme 5)

These processes were further subdivided by Chopra *et al*[37] into those reactions involving radicals (Scheme 4) and those involving carbenes (Scheme 5). Whilst the processes are feasible, in that they do lead to the correct products, there is no evidence for the selectivity of the reactions. With such a wide variety of radicals present one might expect a wider range of products than those detected experimentally. For the proposed mechanism to be correct many of the reactions, especially radical - radical combinations, would have to proceed much faster than certain other processes that do not seem to differ in any major way. A great deal of evidence would have to be obtained to support such a case and, indeed, the building of a consistent explanation for the success of some radical reactions over others is by no means assured.



Scheme 4 Mechanisms involving the fragmentation of DDT to produce radicals.

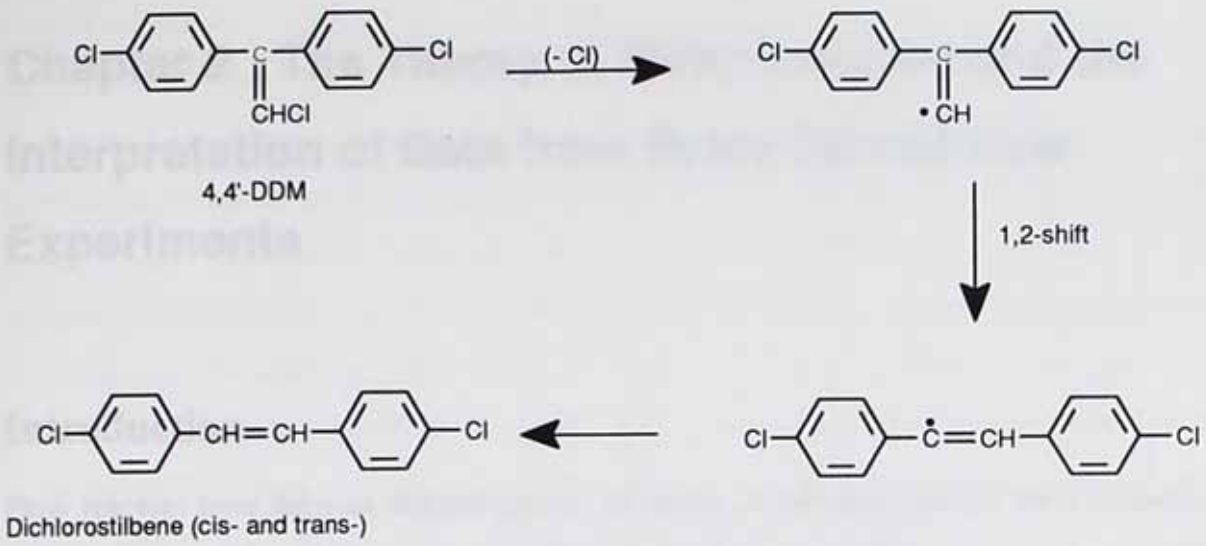
However, it is possible that some of the reactions included in the mechanism are correct. A further investigation of the mechanism might start by using this reaction set. One avenue worth considering is that a radical chain mechanism might exist and be responsible for many of the products detected. In such a case the radical combinations and the hydrogenation / hydrochlorination of the radicals proposed would probably not be very important. A chain mechanism is often composed of a few very important reactions, e.g. abstractions or bond homolyses, that are repeated in the propagation of the chain.



Scheme 5 Mechanisms involving the formation of carbenes

Mechanisms involving rearrangement reactions (Scheme 6)

Such a reaction sequence as that shown in Scheme 6 seems to be a valid explanation of the production of dichlorostilbenes in the pyrolysis. Again, however, this is merely a hypothesis and more work needs to be done to ascertain the extent to which it is true. In contrast to the radical mechanisms above (Scheme 4 and Scheme 5), there are no significant inconsistencies apparent in the mechanism.



Scheme 6 Mechanism involving a rearrangement step

The case of the burning cigarette is, as mentioned above, much different to the situation in an inert atmosphere due to the larger number of H atoms available for reactions in the former environment. One can speculate that the atmosphere in an incinerator would be much more oxidising than this since care is taken to ensure that 6% excess oxygen is present in order to avoid the formation of product of incomplete combustion (PIC). However, the study above shows the usefulness of an investigation in an inert gas. The comparison between the inert gas situation and the tobacco smoke pyrolysis has allowed the authors to draw conclusions by making predictions based on the known differences in the atmospheres and judging the similarity between theory and experiment. In addition, they were able to obtain information about likely processes in the more complicated tobacco burning experiment from the controlled nitrogen pyrolysis.

Chapter 2 : The Theory of Flow Reactors and the Interpretation of Data from Pulse Stirred-flow Experiments

Introduction

Flow reactors have been an integral part of the study of chemical kinetics for a considerable period of time. The technique is designed to transpose one reaction parameter that is difficult to measure or control, i.e. in this case, time, to one that is more easily manipulated. The alternative quantity most often chosen is the velocity of the reacting fluids through the reaction system. For example, in a tube reactor the concentration of reactants and products is invariant with time at a particular point along the reactor. There exists a differential concentration from the reactor entrance to the exit. In this way the experimentalist attains the ability to vary the 'window of time' being scrutinised from several thousand seconds to the order of milliseconds[38] as flow is varied.

Flow techniques can be divided into three categories: linear flow, stirred-flow and pulsed stirred-flow. Linear flow is the simplest of the group but, for several reasons examined in more detail below, is unsuitable for complex reactions. For this reason, most use of linear flow is reserved for those reactions exhibiting first or pseudo-first order behaviour. For complex reactions linear flow methods have been superseded by stirred-flow techniques. One of the greatest remaining disadvantages of the stirred-flow technique, the need to use large amounts of materials, has been overcome through the use of pulsed stirred-flow technology. However, although the chronological development of each flow method was driven by the need to overcome the disadvantages of each previous method, none of the flow methods is perfect, each having its own problems in different circumstances.[39]

Linear Flow

In a typical experiment two streams of reactant gas are mixed at the beginning of a heated reaction zone and the products analysed downstream of the reactor (Figure 5). The reaction zone may take the form of a tube approximately 2-3 cm in diameter and 20-30 cm in length (longer, coiled tubes are employed occasionally to work on slower reactions). As the duration of reaction is governed (or assumed to be governed) by the amount of time spent in the reaction zone, a range of times can be scrutinised by varying the flow rate. Usually, for work at atmospheric pressure, the reactants are mixed with a large fraction of inert carrier gas (in order to reduce the amounts of reactants needed and to minimise local heating or cooling likely to take place in the presence of large quantities of reacting gas). However, much of the work performed using this technique has been done at pressures of 660-6600 Pa, utilising no carrier gas.

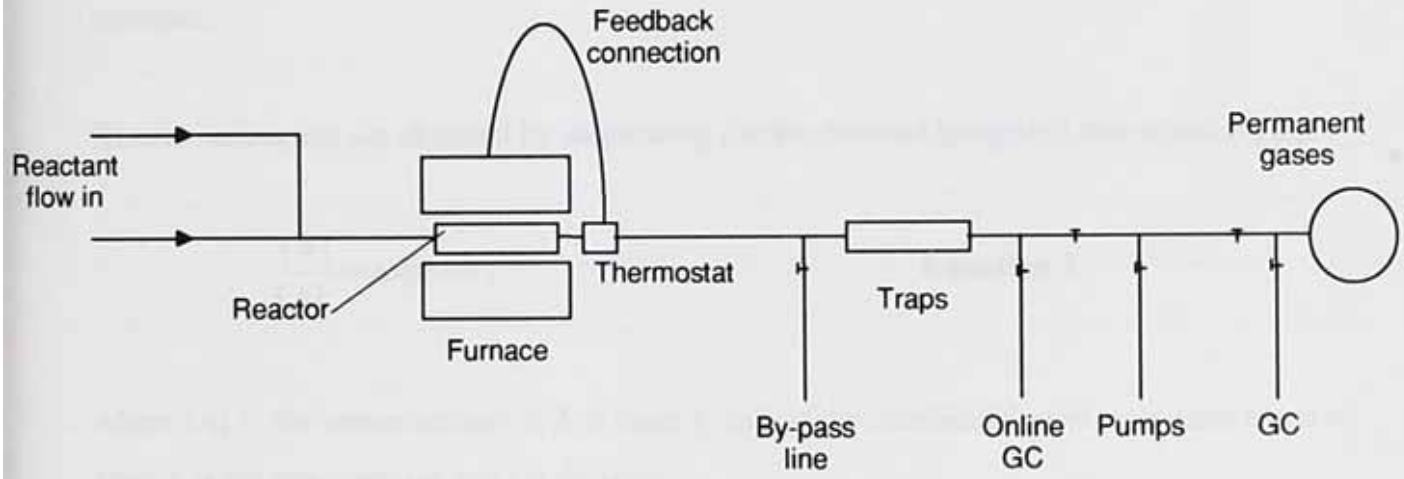


Figure 5 Simplified diagram of a flow-tube apparatus[40]

A range of analysis methods exist but those most frequently employed are gas chromatography (GC) and spectroscopy, e.g. infra-red spectroscopy. Spectroscopy is advantageous, if applicable, as analysis can be arranged at a point very close to the exit from the reaction zone, thereby fixing the reaction time explicitly. However, spectrometric methods rely on the material being measured having a suitable, characteristic absorption. More common than

spectrometric methods is the practice of collecting the products for a measured amount of time and their subsequent analysis by GC.

Kinetic parameters

Kinetic parameters depend largely on the residence time in the reaction zone, usually assumed to correspond to the reactor tube. In such a case, the residence time (t) is given by[39] :

$$t = \frac{PV}{RT\sum_i N_i} \qquad \textbf{Equation 1}$$

Where t is the residence time, P, V and T the pressure, volume and temperature of the reactor, respectively. $\sum_i N_i$ is the total number of moles entering the reactor in unit time, R is the gas constant.

Kinetic parameters are obtained by substituting t in the relevant integrated rate equation, e.g. :

$$\frac{[A]}{[A]_0} = \exp(-kt) \qquad \textbf{Equation 2}$$

where [A] is the concentration of A at time, t, $[A]_0$ is the concentration of A at time equal to zero, k is the rate constant and t is the time.

Limitations

There are, however, several disadvantages with tubular flow methods.[39] It can be very difficult to define the reaction time with distance relationship and the reaction temperature. One of the assumptions inherent in the technique is that the gas attains the reaction temperature immediately on entering the reaction zone and that the reaction is instantaneously quenched on leaving. Although these assumptions are never completely true, better approximation is obtained by increasing the length of the reaction zone and decreasing the flow rate. However,

decreased flow rate allows back diffusion of products and intermediates against the direction of flow. Thus control of the reaction time is lost. This is especially the case if free radicals or other reactive intermediates escape from the reaction zone, either continuing the reaction past the desired quench point or initiating it too soon. Such problems are accentuated by the pressure drop along the reactor. Higher order processes (i.e. greater than 1) occur faster at the entrance than at the exit. In addition temperature differences can occur locally from the heat liberated or absorbed by the reaction. If the extent of reaction is sufficient or the concentration of reactants high enough, serious local heating effects can occur, radically changing the kinetics of the process.

Such scenarios are of less importance if instantaneous, optical / spectrometric methods are used to analyse the reaction mixture. However, it is common, especially with gas-phase systems, to use gas chromatography as the analytical technique. This requires a well defined and well known reaction zone / time.

Stirred-flow

Stirred-flow techniques are very similar to linear flow techniques in many ways. A flow of reagent, usually sustained in a carrier gas, is maintained through a heated reactor in which a steady state eventually comes to dominate. However, there are two assumptions made that contrast with those made for linear flow methods and are the source of many of the advantages of this technique. They are:

a) Mixing is assumed to be uniform throughout the reactor volume

b) The mixture flowing out of the reactor is assumed to have the same composition as that within the reactor

Assumption a) represents a great improvement over the equivalent premise for linear flow, that there is no mixing within the reactor. As diffusion interpenetration always occurs, it is impossible to reduce the degree of mixing below a certain level whilst it remains relatively simple to increase artificially the mixing through the use of a suitable device, such as a reactor designed to have a high degree of turbulent flow (Figure 6). Also, since all the species are

present at all points in the reactor in the same concentrations when a steady state prevails, there is no risk of differential heating due to heats of reaction.

To ensure complete mixing reactor design is important. The stirred-flow reactor used by Mulcahy and Williams[41] (Figure 6) ensures mixing by the forced convection resulting from the gas flow from the perforated sphere in the centre of the vessel.

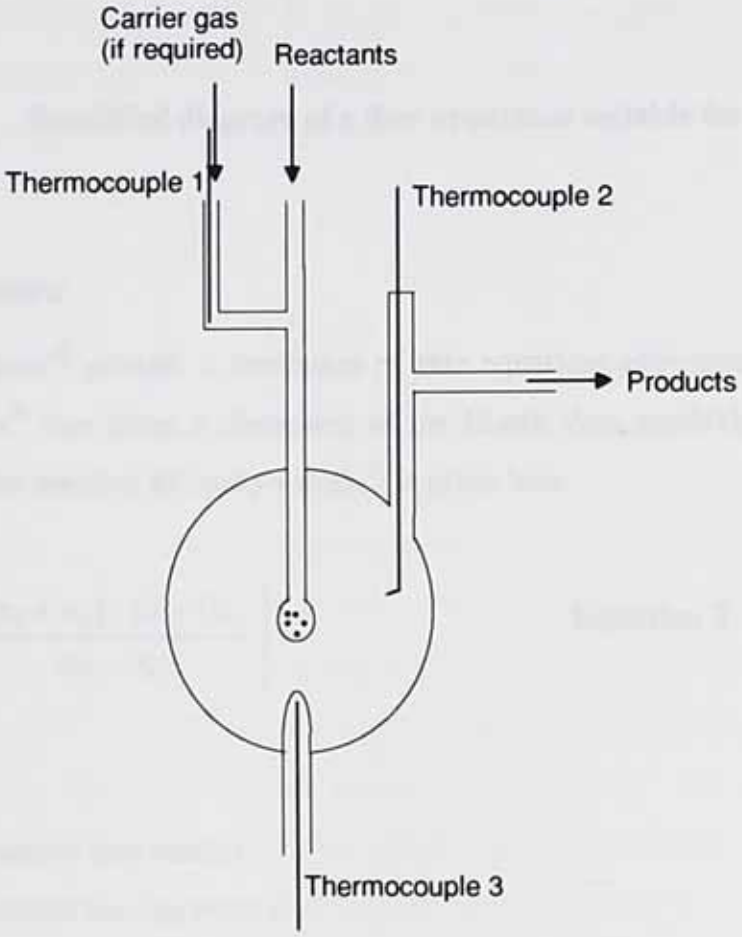


Figure 6 **Diagram of a stirred-flow reactor used by Mulcahy *et al*[41] to investigate the decomposition of di-*t*-butyl peroxide**

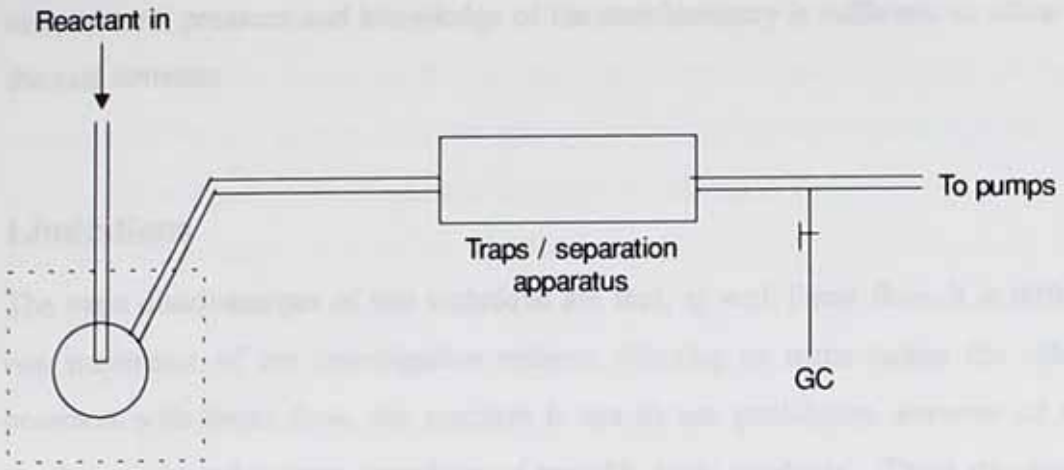


Figure 7 Simplified diagram of a flow apparatus suitable for stirred-flow

Kinetic parameters

Mulcahy and Williams[41] provide a derivation of rate equations appropriate for stirred-flow reactors. Herndon[38] also gives a discussion of the kinetic data available from stirred-flow techniques. Only the result of Mulcahy's work,[41] is given here.

$$k = \frac{n_p RT}{xPV}\left[\frac{x(n_0 + n_c) + (x-1)n_p}{xn_0 - n_p}\right] \qquad \textbf{Equation 3}$$

Where :

n_0 rate of reactant into reactor

n_p rate of product leaving reactor

n_c rate of flow of inert gas through the reactor

x number of product molecules formed by the reaction of one reactant molecule

P total pressure in the reactor

V total volume of the reactor

T temperature in the reactor (K)

k first order rate constant

The main point of interest is the extreme facility with which kinetic parameters are obtained from stirred-flow experiments. One simply needs to measure the flow of the gas leaving and entering the reactor, determine the concentrations of the reagents and this, together with the

temperature, pressure and knowledge of the stoichiometry is sufficient to allow calculation of the rate constant.

Limitations

The main disadvantages of this technique are that, as with linear flow, it is difficult to change one parameter of the investigation without effecting to some extent the others. Also, in common with linear flow, the reaction is apt to use prohibitive amounts of reagents or to produce undesirably large quantities of possibly toxic products. These disadvantages can be overcome through the use of the pulse stirred-flow technique.

Pulse Stirred-flow

The pulse stirred-flow technique was pioneered by Davidson *et al*[42,43] for the investigation of the gas-phase kinetics of organosilicon compounds. It combines the simplicity of stirred-flow with greatly reduced requirements in terms of sample size. As originally designed, the technique involved the use of a standard packed column gas-chromatograph (GC) equipped with a gas sampling valve (GSV). Inserted in the carrier gas line between the GSV and the GC was a stirred-flow reactor capable of being mounted in a tube furnace for heating. Attached to the GSV was a vacuum line from which low pressures of reagent gas could be injected into the carrier gas and, hence, flow through the reactor (Figure 8). The essential difference between this and normal stirred-flow techniques is that, instead of a continuous stream of reactant through the reactor, single analytically sized samples are injected.

It is, of course, integral to the technique that the amounts of reactants and products leaving the reactor can be measured by the GC. Further modifications to the set-up were undertaken to improve the measurement of the compounds in the reactor effluent. (Also, it is necessary to know how much reactant has been injected into the system. Usually this is done through knowing the volume of the sample loop of the GSV exposed to the reactant gas in the vacuum line and by measuring the pressure of the gas in that loop).

The packed column GC was changed to capillary, a measure which necessitated additional modifications to decrease the peak width of some of the least retained materials. As detailed

below, the loss of reactant and product from the reactor after the initial pulse is a first order process so that the concentration in the effluent gas stream of any component follows an exponential decay. Thus it can take some time for the majority of the material to have left the reactor. (Davidson *et al*[42,43] quote a value of *ca.* 5τ (where τ is the residence time - see below) for 99% of the pulse to have left the reactor). This results in a very badly tailed peak for compounds that have retention times < 10τ. In order to avoid this problem, a cryogenic focusing technique is used. A trap inserted in the gas-line between the reactor and the GC is cooled, e.g. with liquid nitrogen, until all the material has left the reactor and been collected there. The trap is rapidly heated, e.g. in boiling water, volatilising all the materials into the gas stream and thus into the GC as a concentrated pulse. In addition, it is necessary that the GC be equipped with a non-discriminating splitting device to reduce the flow of gas to that appropriate for a capillary column.

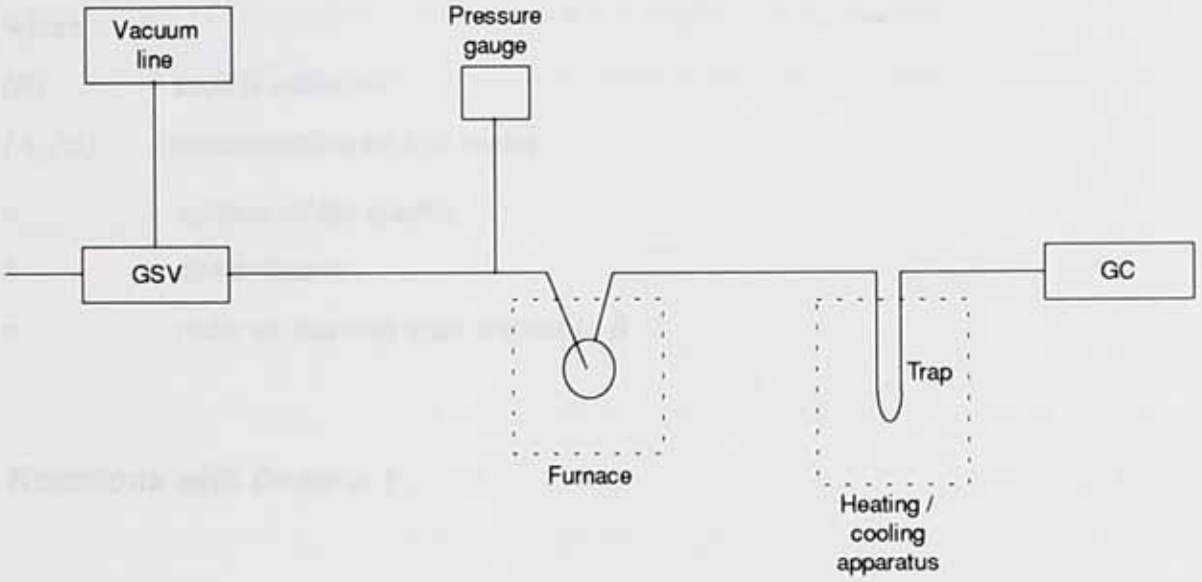


Figure 8 Schematic diagram of a pulse stirred-flow apparatus

Kinetic Parameters

Although pulse stirred-flow has much in common with stirred-flow technology, the use of a finite pulse requires significant modification to the equations necessary to obtain kinetic information from the technique. As well as the assumptions already made for normal stirred-

flow, i.e. that mixing is assumed to be uniform throughout the reactor volume and that the mixture flowing out of the reactor is assumed to have the same composition as that within the reactor, another is necessary for the pulse method. It is assumed that 'perfect pulse behaviour' is in effect, i.e. that the volume of the pulse is so small relative to the reactor that it enters and mixes throughout the reactor almost instantaneously.

In the case of a single reactant being injected, the amount of any product formed within the reactor is the same as the amount that is collected at the outlet. Since the mixing in the reactor is assumed to be uniform, the material at the outlet at any time will represent the composition of the contents of the reactor. Thus :

$$(B) = \int_0^\infty kv\left[A_o(t)\right]^n dt \qquad \textbf{Equation 4}$$

where :

(B)	total B collected
$[A_O(t)]$	concentration of A at outlet
v	volume of the reactor
k	rate constant
n	order of reaction with respect to B

Reactions with Order = 1

For a first order reaction, $n = 1$, the solution to Equation 4 is exact. If (A_O) is taken as the total amount of A collected at the outlet then:

$$(A_o) = \int_0^\infty u\left[A_o(t)\right]dt \qquad \textbf{Equation 5}$$

where u is the volumetric flow rate.

For constant u :

$$\frac{(A_o)}{u} = \int_0^{\infty} [A_o(t)]dt \qquad \textbf{Equation 6}$$

which can be substituted in Equation 4 to give:

$$k\frac{v}{u}(A_o) = (B) \qquad \textbf{Equation 7}$$

which rearranges for k as :

$$k = \frac{u(B)}{v(A_o)} = \frac{1}{\tau}\cdot\frac{(B)}{(A_o)} \qquad \textbf{Equation 8}$$

where τ is the residence time. Equation 8 is true regardless of the extent of reaction as long as there is perfect mixing in the reactor and the order of reaction, n, is unity.

Reactions with Order ≠ 1

To solve Equation 4 for the amount of *(B)* collected, it is necessary to know the time dependence of $[A_O]$ exactly. If n ≠ 1, the time dependence due to reaction is not known. Therefore, one must use a small extent of reaction and assume the concentration of *A* in the reactor (and hence at the outlet) depends on the 'sweeping out' process alone. In order for this to be of use, the time dependence of the sweeping out process must be known. Therefore, it is assumed that the sweeping out is a first order process given by :

$$[A_o(t)] = \frac{(A_o)}{v}\exp\left(\frac{-t}{\tau}\right) \qquad \textbf{Equation 9}$$

This is true only if the perfect pulse assumption holds. Substitution of Equation 9 in Equation 4 gives:

$$k = \frac{(B)v^{(n-1)}n}{(A_o)^n \tau}$$ **Equation 10**

Rearrangement of Equation 10 gives :

$$(B) = (A_o)^n \frac{k\tau}{v^{(n-1)}n}$$ **Equation 11**

Therefore a graph of log*(B)* against log*(A_O)* has slope *n*. From the previous equation it can be seen the vital role that the value assigned to the order has in determining the rate constant. It follows that the reliability of the rate constant depends on the quality of the determination of the order.

Davidson *et al*[42,43] have detailed many other modifications and refinements to the technique to deal with other situations, e.g. reversible reactions, competing reactions and order determination of a minor pathway in the presence of a first order major pathway, but they are not described here.

Chapter 3 : Experimental

Chemicals

All chemicals were obtained from Aldrich and were used without further purification, unless explicitly stated.

Handling of air sensitive materials

Air sensitive materials, e.g. 1,1-dichloroethene, were dispensed in small amounts (< 4 cm^3) into a vacuum line sample tube, that had previously been evacuated, in a nitrogen filled glove-bag. Positive pressure of nitrogen sufficient to inflate the bags to almost their full volume was used throughout the entire dispensing procedure. Prior to use the bags were flushed with nitrogen at least twice.

Having been dispensed into the sample tube, the chemicals (always liquids) were degassed by repeated freezing in liquid nitrogen, evacuation of the tube, disconnection from the vacuum and slow warming to room temperature. This was performed until no further bubbling of gas from the liquid was observed under vacuum. Finally, the sample was vacuum distilled into another sample tube whilst continuous pumping was applied (see the section on the Vacuum Line for more detail of the distillation procedure).

Overview of the reaction and analysis system for the pulse stirred-flow technique

The apparatus required for this technique is, essentially, quite simple. Indeed, that is one of the factors that influenced the choice of methodology for this work. All that is necessary to carry out a kinetic investigation is a gas chromatograph (GC), reactor with controllable temperature and gas flow, and a method of handling and purifying small quantities of gaseous materials of which some are air sensitive. The last requirement was met by the use of a vacuum line connected via a gas sampling valve (GSV) to the gas supply flowing through the reactor and

hence into the GC. Thus, a working system would be able to degas and distil, using trap to trap techniques, a liquid sample, possibly quite volatile, deliver a precisely known and small amount into the carrier gas flowing through the reactor and analyse the effluent material quantitatively for a variety of calibrated chemicals.

System Components

Furnace

Design

The furnace was designed to provide a means of obtaining a wide range of temperatures with the minimum of variance and equilibration time. This had to be done while working within the constraints resulting from the decision to modify and rebuild an existing heating system from a similar experiment. Thus, it was necessary to provide a means of insulating the bare wire of the existing heating coil whilst ensuring that the heating effect was controllable and the temperature in the reactor known.

The above aims were achieved by placing the coil, which was wrapped around a steel cylinder within a larger cylinder of a thermal and electrical insulator, inside an earthed, steel case. The steel case was breached by a hole in one side allowing access of the power supply. The power supply was modulated by a microcomputer operated process controller placed in a separate unit (to ensure no heat damage to the delicate electronics). Heating current was controlled by a solid state relay placed in the circuit between the coil and the mains. The state of the relay was switched by a small current from the process controller which took an input from a thermocouple located within the reactor in the furnace allowing temperature monitoring and the possibility of automatic control.

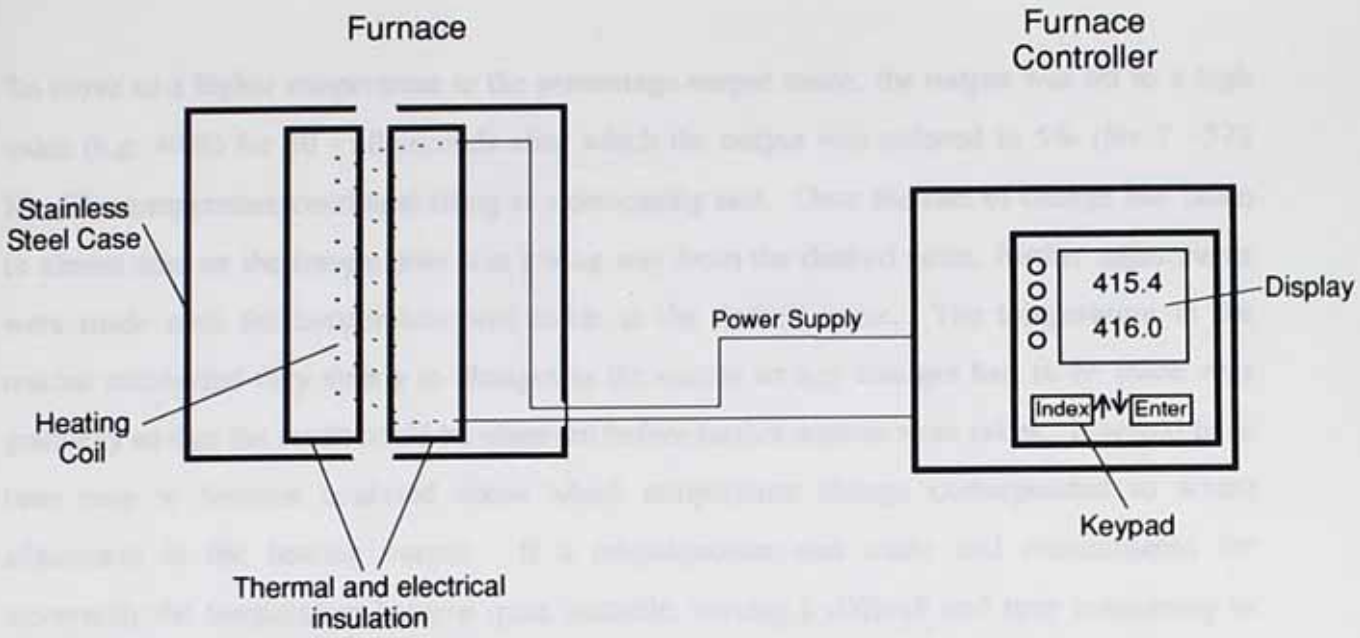


Figure 9 Diagram showing the furnace and controller arrangement

Procedure of Operation

The controller has two basic modes of operation, an automatic heating mode and a percentage heating mode. In the automatic heating mode the microprocessor uses internal settings to calculate when to heat the furnace and when not to, in an attempt to keep the probe temperature at the specified value. The values used to calculate heating times can be either preset for slow, medium or fast processes, or learned by the controller. Due to the nature of this furnace, however, it was particularly hard for the microprocessor to adjust to the process and the reactor temperature does not remain very stable. Therefore it is better to use the other mode of operation, the percentage output.

When in the percentage output mode, the microprocessor divides time up into pre-determined slices and turns the furnace on for a percentage of every slice. The time slices can be varied from 2 to 8 seconds. The shorter the time slice setting the more even the heating effect. The problem with this mode is that it is a matter of trial and error to find the correct percentage setting for a particular temperature. For temperatures up to 970 K the output needed is unlikely to be more than 20%. Once the correct value has been found it is possible, with small adjustments, to keep the temperature stable ± 0.3 K.

To move to a higher temperature in the percentage output mode, the output was set to a high value (e.g. 40%) for 30 - 60 seconds after which the output was reduced to 5% (for T ~570 K). The temperature continued rising at a decreasing rate. Once the rate of change had fallen to almost zero or the temperature was a long way from the desired value, further adjustments were made until the temperature was stable at the desired value. The temperature in the reactor responded very slowly to changes in the output so any changes had to be made very gradually so that the result could be observed before further actions were taken. It would have been easy to become confused about which temperature change corresponded to which adjustment in the heating output. If a misjudgement was made and compensated for incorrectly the temperature became quite unstable, making it difficult and time consuming to set the temperature to the desired value.

When using the procedure outlined above, it was necessary to use some caution in setting the output. It was important that the output was not left unattended on a high setting (>15%) as there is no upper temperature limit on the furnace in this mode. Heating continues until the system reaches a natural equilibrium with its surroundings. The equilibrium temperature for a reasonably high setting (>20%) is likely to be greater than the desired, or even the safe, operating temperature of the equipment in the furnace.

Reproducibility

Extensive tests over a period of several hours were performed to determine the extent to which the temperature of the furnace could be maintained at a given level. As mentioned in the procedure above, the percentage output method of control was found to be much more effective in maintaining a highly accurate degree of control over the temperature than the fully automatic mode. The difficulty in maintaining good temperature control during automatic mode arose from the lag between adjustments made by the controller and their effect on the temperature measured by the probe, i.e. the process controller was unable to adjust sufficiently to the sluggishness of the process. The delay in the response of the furnace was due to the large mass of the furnace coil together with the fact that the reactor was insulated from the walls by air.

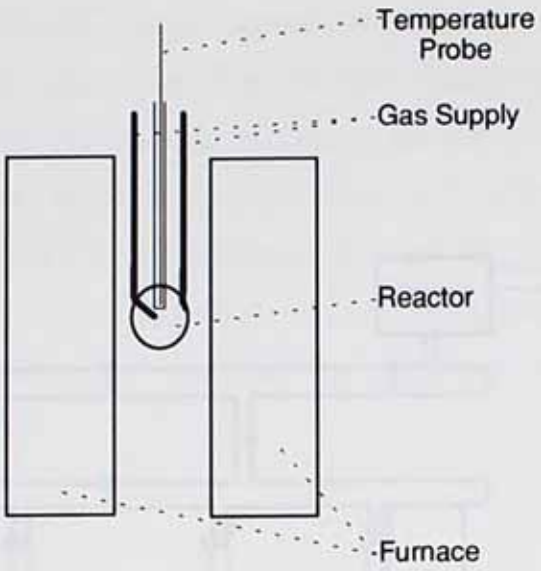


Figure 10 Diagram depicting the relative position of the temperature probe and the furnace

With careful control, the furnace temperature was found to be reproducible within the range 670-870 K with a fluctuation of 0.3 K. The phrase 'careful control' refers to the necessity of monitoring the temperature in the reactor every 15 minutes and adjusting the heating output slightly in the case of any change in temperature. This procedure proved to be easy to follow and caused no substantial problems.

Vacuum Line

Design

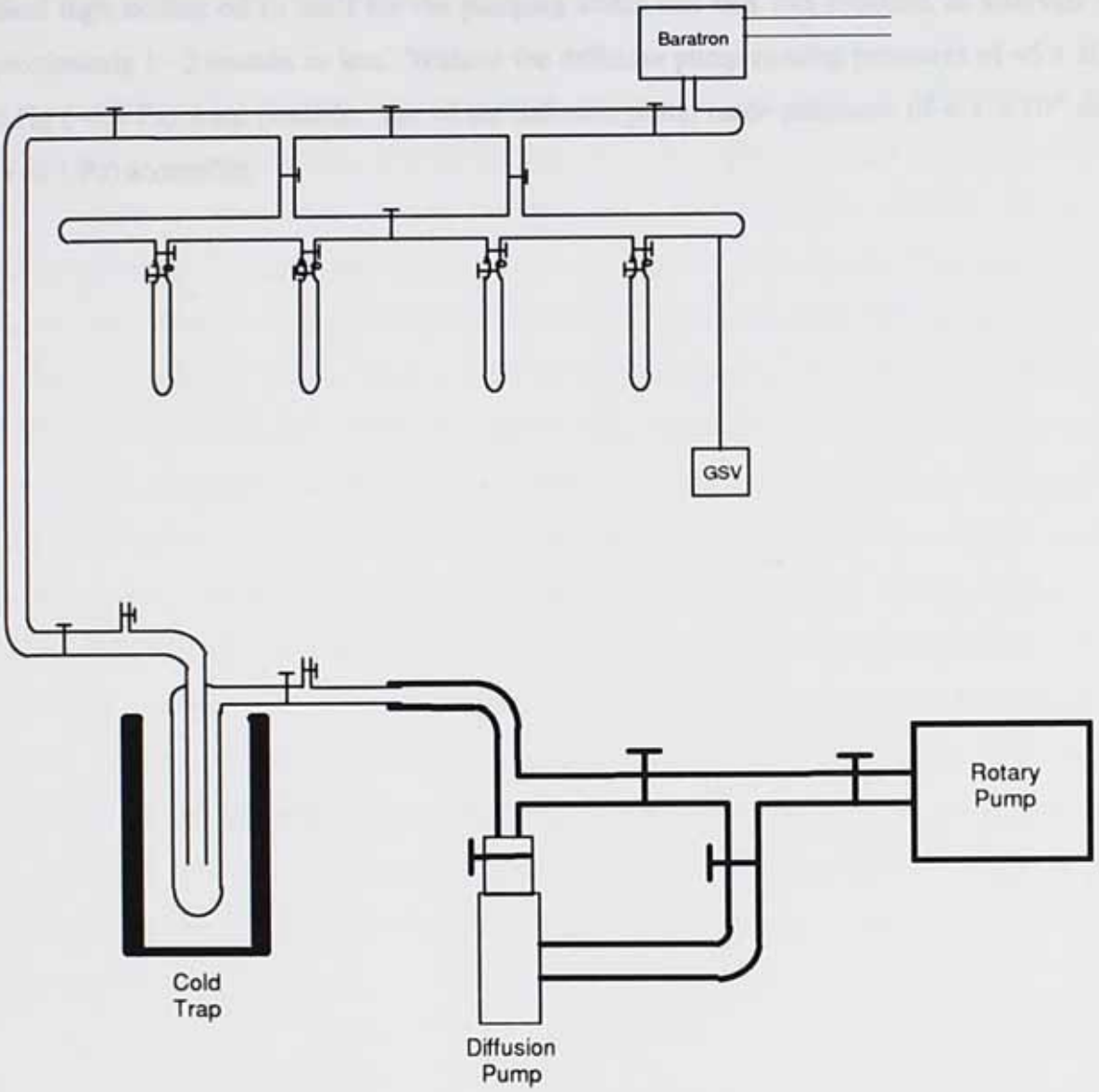


Figure 11 **Diagram of the vacuum line showing position of pumps, valves, stopcocks, pressure gauge (baratron) and sample tubes**

The vacuum line was custom built* to be flexible in function, easy to use and requiring the minimum of maintenance or repair in the event of a breakage. The sample tubes were

---

* For which the co-operation of the Chemistry Department of Leicester University is acknowledged gratefully.

equipped with threaded fittings that allowed secure attachment to the vacuum line or the fitting of plastic caps affording protection from dust and from damage to the delicate glass of the vacuum fitting. All the other fittings were of Youngs greaseless variety. The baratron was connected to a digital readout with a range of 0-11 mm Hg (0-1460 Pa). The diffusion pump utilised high boiling oil (6 cm^3) for the pumping effect and this was replaced at intervals of approximately 1 - 5 months or less. Without the diffusion pump running pressures of ~5 x 10^{-2} mm Hg (~6.5 Pa) were possible; use of the diffusion pump made pressures of < 1 x 10^{-3} mm Hg (~0.1 Pa) accessible.

Gas Chromatography

Gas chromatography was performed using a Perkin Elmer 8500 gas chromatograph equipped with a 25 m long column with internal diameter of 0.22 mm, film thickness of 0.11 μm and stationary phase dimethyl-polysiloxane gum.

Given the electrophilic nature of many of the materials of interest in the investigation, it had been planned to use an electron capture detector (ECD) in conjunction with a flame ionisation detector (FID). The ECD is very sensitive to polyhalogenated materials (such as trichloroethane [TCE]) and very selective to these. Indeed, detection limits as low as 10^{-19} moles of some materials have been achieved.[44] The FID complements the ECD well in that it is sensitive, although not to the same degree for polyhalogenated materials, and capable of detecting virtually all organic compounds. In this way, a possible experimental scenario would be to perform a pyrolysis and analyse the products with both detectors. All organic products would be detected by the FID and the normal methods of retention time comparison and standard addition could be used to identify them. However, the ECD would enhance this process of identification by selectively picking out the highly polar compounds such as TCE. Quantitative analysis could be carried out in the same way. The information from the FID would be adequate in most instances but the extra sensitivity of the ECD would increase the accuracy of data on the polar compounds, likely to be of most interest as the environmentally most influential materials. A primary objective, therefore, of the preliminary stages of this work was to set-up the gas chromatograph and ensure the operation and reproducibility of both detectors.

Flame Ionisation Detector (FID)

(See Appendix A for an explanation of the working principles of the FID).

No serious problems were encountered when setting up the FID. These detectors are known to be reliable and although not capable of the upper range of the ECD's sensitivity, they are among the most sensitive commonly used detectors. Having set up the gas flows for the air and hydrogen as 450 cm^3 min^{-1} and 45 cm^3 min^{-1} respectively, calibrations for 1,1,1-

trichloroethane (TCE) were carried out (Figure 12). As the response was found to vary little from one day to the next, and in view of the fact that the air and hydrogen flows were turned off each day (requiring them to be reset the next morning), such reproducibility was considered more than adequate.

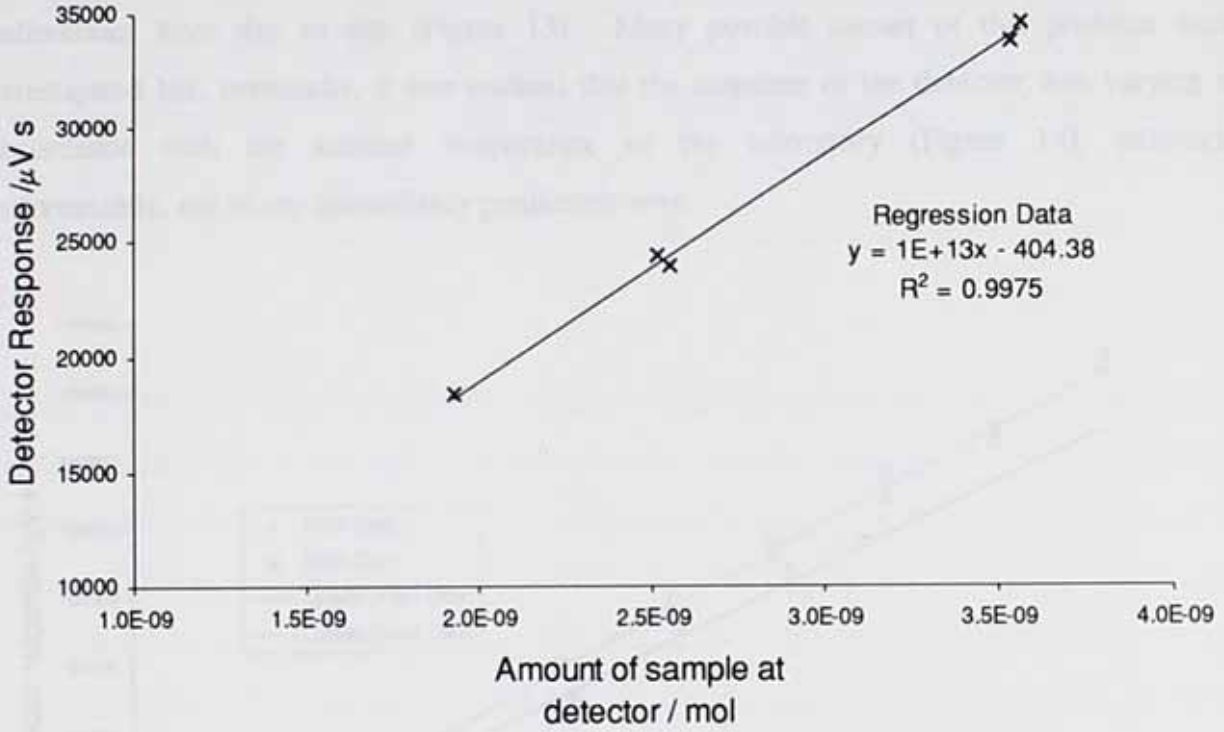


Figure 12 Graph showing a selection of data for the calibration of the FID with 1,1,1-trichloroethane (TCE), together with the linear regression results, collected over a range of temperature (290-295 K); split flow 20.1 ± 0.13 cm^3 min^{-1}, column flow ~0.5 cm^3

Electron Capture Detector (ECD)

(See Appendix A for an explanation of the working principles of the ECD).

On preliminary testing with a solution of TCE in hexane (concentration, C, = 3.56 x 10^{-4} mol dm^{-3}), this detector was found to be extremely unreliable, producing widely differing calibrations from day to day (Figure 13). Many possible causes of this problem were investigated but, eventually, it was realised that the response of the detector was varying in accordance with the ambient temperature of the laboratory (Figure 14), although, unfortunately, not in any quantifiably predictable way.

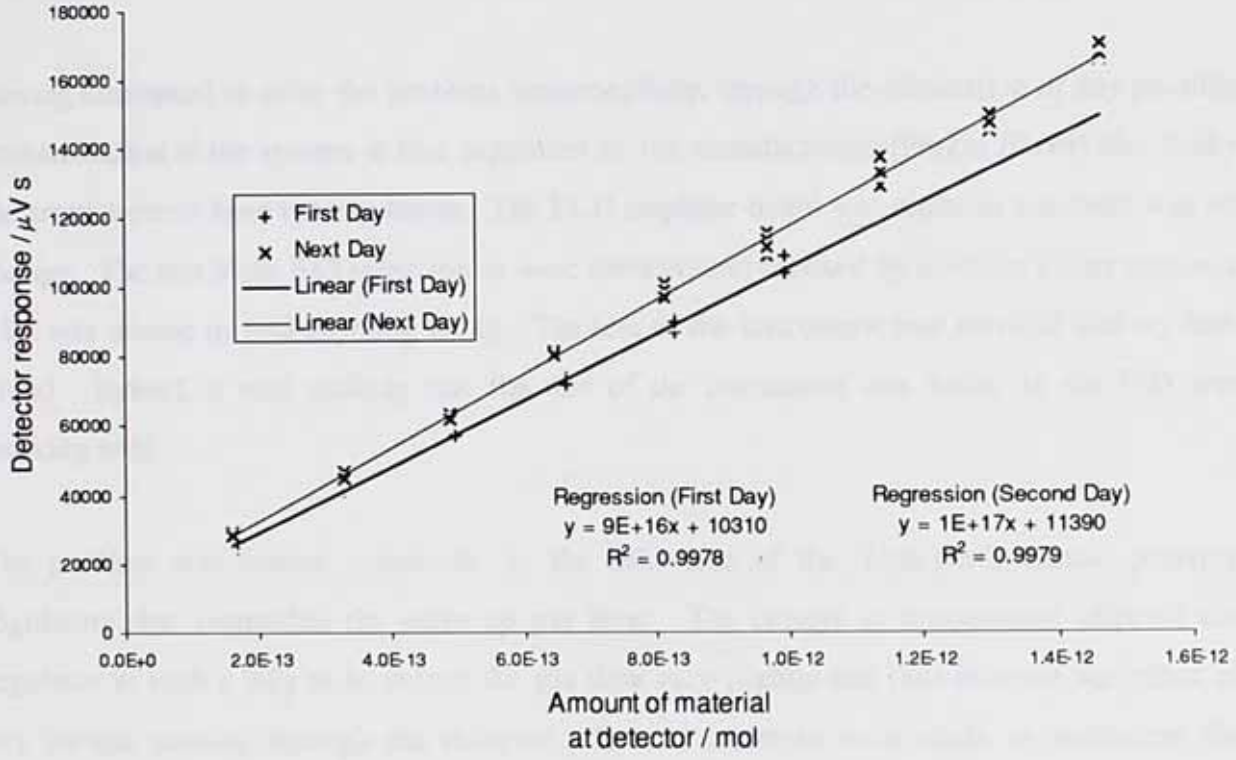


Figure 13 Graph showing the response of the ECD on two consecutive days. Note the divergence of the lines of best fit for each set of data.

At first the temperature dependence of the detector's response was thought to be due to contamination of some sort. ECD's are known to be very sensitive to contamination, especially by electronegative compounds such as water, oxygen or, possibly by highly fluorinated alkanes leached from the teflon tubing originally used in the gas supply system. In

order to test this, the ECD cell was cleaned, both by soaking in hexane and drying, and by heating at 400 °C with a high flow of make-up gas (N_2). The column guide into the ECD was replaced also. A very detailed inspection of the gas supply lines was carried out and all detectable leaks fixed. Thus it was assumed that the detector itself was clean and that the gas supply was not being contaminated with O_2 or H_2O from outside the line (an effect was thought to have been possible despite the oxygen and moisture traps in the supply). As an extra precaution, the moisture and oxygen traps were renewed. Finally, there being no improvement after these measures were taken, the entire gas supply system was replaced with new stainless steel tubing (particularly the parts that had been teflon tubing) and the whole leak testing procedure repeated. However, there was no great improvement in the detector response.

Having attempted to solve the problem, unsuccessfully, through the elimination of any possible contamination in the system, it was suggested by the manufacturers (Perkin Elmer) that faulty electronics could have been to blame. The ECD amplifier board was replaced but there was no change. The rest if the ECD electronics were serviced and checked by a Perkin Elmer engineer who was unable to find anything faulty. The rest of the instrument was serviced and no fault found. Indeed, it was unlikely that the rest of the instrument was faulty as the FID was working well.

The problem was traced, eventually, to the behaviour of the 'high-performance' pressure regulators that controlled the make-up gas flow. The change in temperature effected the regulator in such a way as to reduce the gas flow very slightly and thus increase the effect of any sample passing through the detector. Several attempts were made to overcome the sensitivity of the regulators to temperature through exchanging them with other models containing diaphragms made of stainless steel. Unfortunately, this did not reduce the effect sufficiently as there was still a discernible effect, even within the same day, if the temperature changed by even as much as 5 K (as it did frequently in summer).

The only possible solution was either to keep the temperature in the laboratory constant at all times or to use an internal standard. The laboratory was not equipped to be air conditioned so temperature control was impossible. It was decided not to use an internal standard as it would have had to have been added to the reactant before injection, perhaps effecting the

mechanisms of pyrolysis in an unknown way. Such an unknown effect would have invalidated any work performed under those conditions. It was concluded that the solution to the irreproducibility of the ECD's results was beyond the means available and that the investigative work would have to be performed solely using the FID.

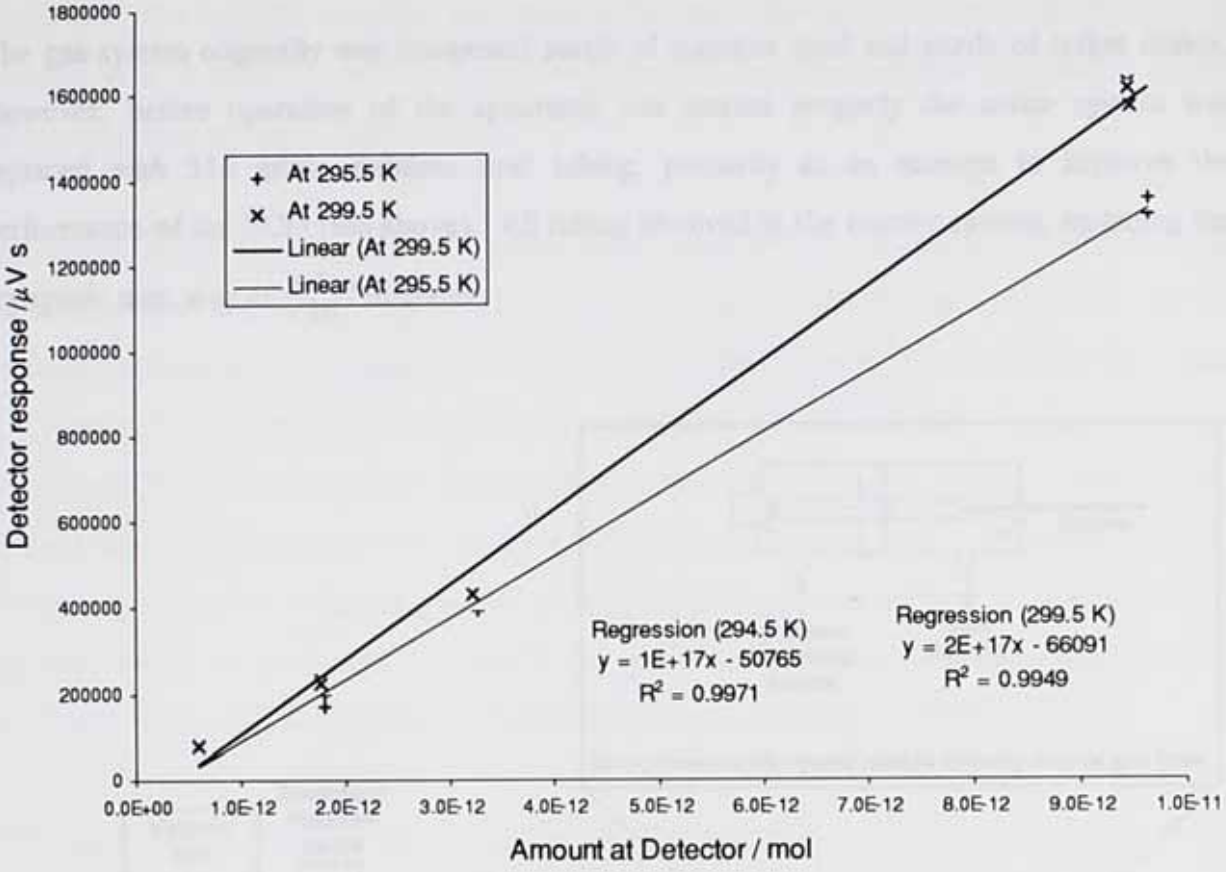


Figure 14 Graph showing the effect of temperature on the response of the ECD to 1,1,1-trichloroethane

Gas System

Overview

The gas-system originally was comprised partly of stainless steel and partly of teflon tubing. However, before operation of the apparatus was started properly the entire system was replaced with 316 grade stainless steel tubing, primarily as an attempt to improve the performance of the ECD (see above). All tubing involved in the reactor system, including the cryogenic trap, was of $1/16$" diameter.

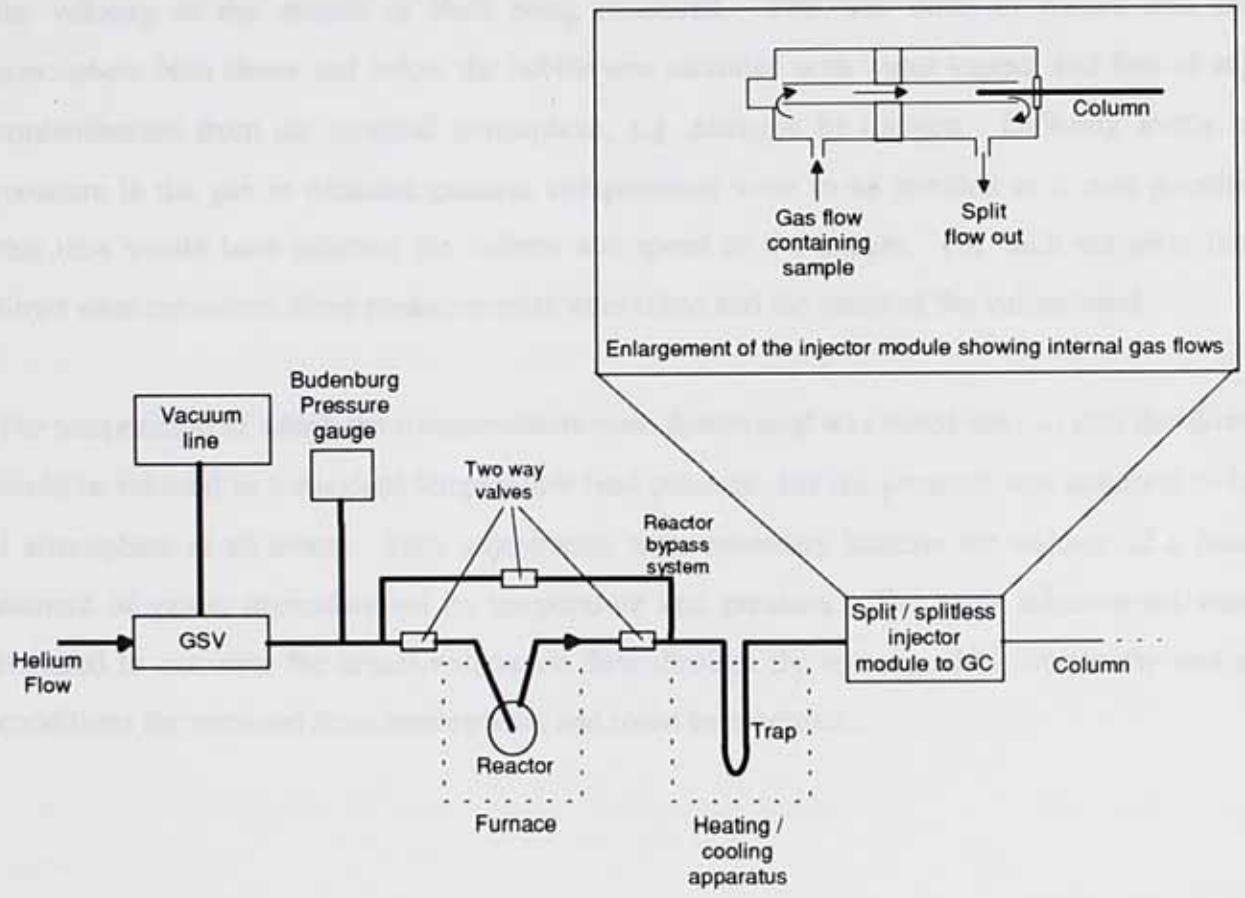


Figure 15 Schematic diagram of a pulse stirred-flow apparatus

The carrier gas used was helium and its flow was controlled by a combination of pressure regulation at the supply and by the combined effect of the split flow setting and the internal bore of the column, i.e. by the flows through the split and column.

Flow measurement

It was necessary to know the column flow, the split flow and the flow through the reactor. Due to the nature of the gas line (see above), the flow through the reactor (in mass terms) was the same as the sum of the split and column flows. Thus, only the column and split flows were measured. The split flow reading was taken at the split flow outlet, the column flow at the detector with all detector make-up and flame gases off. Measurement was done with a bubble flow meter using the procedure below.

Before measuring the flow, several bubbles were passed up the flow tube to ensure that the same level of moisture and, therefore, friction were associated with the tube walls. Actual measurement involved causing at least two bubbles, preferably three, to travel along the tube, the velocity of the second or third being measured. This was done to ensure that the atmosphere both above and below the bubble was saturated with water vapour and free of any contamination from the external atmosphere, e.g. nitrogen or oxygen. Differing levels of moisture in the gas or different gaseous compositions were to be avoided as it was possible that they would have effected the volume and speed of the bubble. For each occasion that flows were measured, three measurements were taken and the mean of the values used.

The temperature at which the measurements were determined was noted also so that the flows could be referred to a standard temperature (and pressure, but the pressure was assumed to be 1 atmosphere at all times). Such adjustments were necessary because the volume of a fixed amount of gas is dependent on its temperature and pressure. The same adjustments were required to calculate the actual volumetric flow through the reactor which frequently was at conditions far removed from atmospheric and room temperature.

Calculation of the flow at reactor conditions

The procedure for calculating the actual flow at the conditions in the reactor is identical to that used for adjusting the bubble meter value for a particular standard temperature and pressure.

Given that :

$$PV = nRT \qquad \textbf{Equation 12}$$

where P is the pressure of a gas, V is the volume of gas, T is the temperature of the gas, n is the number of moles of the gas and R is the gas constant. For constant n, i.e. in this case assuming no gas is lost through leaks :

$$\frac{P_r V_r}{T_r} = nR = \frac{P_a V_a}{T_a} \qquad \textbf{Equation 13}$$

where P_r, V_r and T_r refer to n amount of gas at the reactor conditions and P_a, V_a and T_a refer to n amount of gas at the atmospheric conditions at which the measurement was made. If this calculation were being performed to standardise a flow reading, P_r, V_r and T_r would refer to the standard conditions.

As,

$$V = U \cdot t \qquad \textbf{Equation 14}$$

where U is the volumetric flow rate and t is time, for unit time $t = 1$, $U = V$. Substitution into Equation 13 gives

$$\frac{P_r U_r}{T_r} = \frac{P_a U_a}{T_a} \qquad \textbf{Equation 15}$$

Hence, rearrangement gives the adjusted flow, U_r, as :

$$U_r = \frac{T_r P_a U_a}{T_a P_r} \qquad \textbf{Equation 16}$$

Testing of the 'Perfect Pulse' Behaviour Assumption

The Pulse Stirred-Flow technique requires that 'perfect-pulse' behaviour holds true for the reactor (see Chapter 2 for a discussion of the Pulse Stirred-Flow technique). This is necessary because the concentrations of materials in the reactor and the amounts collected at the outlet are related by :

$$\int_0^\infty kv\left[A_O(t)\right]^n dt = (B) \qquad \textbf{Equation 17}$$

where :

(B)	total B collected
$[A_O(t)]$	concentration of A at outlet at time t
v	volume of the reactor
k	rate constant
n	order of reaction for a reaction of the type : $A \rightarrow B$

In order to solve Equation 17 it is necessary to know $[A_o(t)]^n$. For a first order reaction this is known exactly and the solution of Equation 17 is exact. However, for orders of reaction other than 1, the exact time dependence of the concentration of A is unknown. To overcome this problem, a small extent of reaction is used so that the change in $[A_o]$ is well approximated by the sweeping out of the reactor by the gas stream. The sweeping out of the reactor is assumed to be governed by the relationship :

$$\left[A_o(t)\right] = \frac{(A_o)}{v} \cdot e^{\frac{-t}{\tau}} \qquad \textbf{Equation 18}$$

where :

(A_o) amount of A collected at the outlet

τ the time constant for the reactor, defined as $\frac{v}{u}$ where u is the volumetric flow rate through the reactor

This assumption is true only if there is 'perfect pulse' behaviour. Therefore, it was necessary to check that the assumption was true before undertaking to use the reactor for kinetic investigations. To do this, pulses of 1,1,1-trichloroethane were injected through the reactor and into the GC where the detector (FID) response was monitored. No cryogenic focusing was used on these pulses and the reactor was maintained at 120 °C to avoid any condensation inside. See Figure 16 for an example of the data collected (N.B. the detector response shown has been zeroed as the GC adds a false baseline of ~5320 μV to the detector output). An exponential decay has been fitted to the data. Very good correlation is achieved if one models the data from the first 4 points after the maximum to two points before the end. From comparison of the fitted equation (Figure 6) with Equation 18, it can be seen that $\frac{1}{\tau} = 0.0108$. This corresponds to a volumetric flow rate through the reactor (v = 22.94 cm^3) of ~15 cm^3 min^{-1}. The actual, measured flow at 25.5 °C and atmospheric pressure (assumed 1.01325 x 10^5 Pa) was 21.2 cm^3, corresponding to a reactor flow (at 120 °C, and 1.52 x 10^5 Pa) of 18.7 cm^3 min^{-1}. Given the uncertainties in flow measurement, adjustment to reactor conditions, reactor volume measurement and pulse elongation coupled with unknown trapping effects in the gas line between the reactor and the GC (a distance considerably greater than that relevant to cryogenically focused experiments) the agreement in the flow calculated from the 'sweeping out' experiment and that expected from more conventional methods is very good. This result was taken as convincing evidence that the assumption of 'perfect pulse' behaviour was valid.

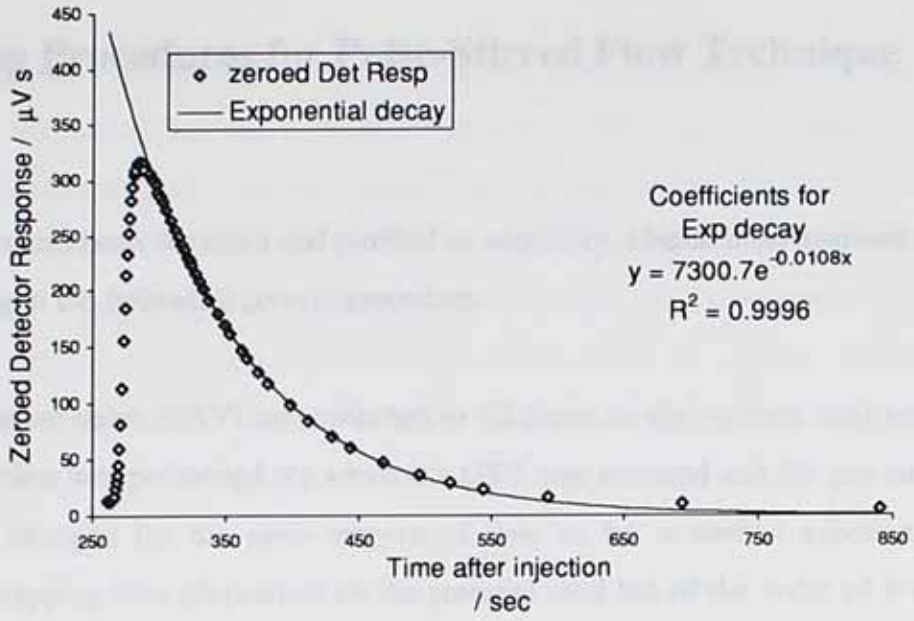


Figure 16 Graph showing the detector response (FID) at a time after injection of a pulse into the reactor (no cryogenic trapping was employed) together with a fitted exponential decay, the coefficients for which are given.

Operating Procedures for Pulse-Stirred Flow Technique

Overview

Once samples had been degassed and purified as necessary, kinetic investigations were carried out according to the following general procedure.

The gas sampling valve (GSV) was switched to fill (open to the vacuum line) and evacuated. A blank injection was performed for which the GSV was actuated and the gas stream trapped using liquid nitrogen for the same amount of time as for a normal injection. After the appropriate trapping time (dependent on the material used but of the order of 8 minutes), the trap was rapidly heated in boiling water and the gas-chromatograph started. If no significant peaks were seen the system was considered clean and the experiment continued. On discovering significant levels of contaminants, however, the source was investigated and the system not used until this procedure showed there to be no contamination.

Having ascertained the system's purity, calibration was performed for the chemicals of interest. Generally, for any particular experiment there were only two materials that needed to be known quantitatively and, therefore, only those two calibrations were carried out. Calibrations were performed via the bypass line, avoiding the necessity of cooling the reactor. (The heating process had to be started sometime before the experiment was performed and frequent, large changes in temperature were considered damaging to the system's integrity and likely to cause leaks so the bypass was essential.) To use the bypass system the valves either side of the reactor were closed and that on the bypass line opened.

Finally, the GSV was switched to fill and, having evacuated the loop, it was filled with a known pressure of the gaseous material to be calibrated, at a known temperature (the room's ambient temperature was used), flushed by the gas stream and trapped using liquid nitrogen as for the blank run. After a set time the trap was heated and the GC started. This procedure was repeated until a satisfactory set of calibration data had been obtained. Due to the reliability of the FID, shown by extensive calibrations over long periods of time, only a few calibration points were used as this allowed more time for the kinetic part of the investigation.

After the calibration had been completed, the reactor was switched back into the gas line and injections of the reagent gas made, as described above. The reactor temperature, split flow, carrier gas pressure and ambient temperature were all recorded at regular, frequent intervals. The reactor temperature was controlled to be as constant as possible (see the Furnace section) while injections were being made. Depending on the aim of the experiment, after several injections of varying amounts, the reactor temperature would be adjusted and the procedure repeated. The results were processed using the program written for this purpose (included in Appendix A) and the output loaded into a spreadsheet for comparison with other data.

Standard addition procedure

On obtaining chromatographic plots of a reaction, preliminary identification would be made from the retention times of the peaks as compared with pure samples of the expected or likely products. Confirmation of this result was sought through the use of a standard addition of the material thought to be giving rise to a particular peak. Having performed one or more runs previously in the normal manner described above, this would be repeated with the standard addition made. Standard additions were made before the injection of the reagent pulse, as for normal injections except that the bypass line was used to convey the material to the cold trap. It was assumed that once in the trap no sample would be lost. Having trapped the standard addition pulse, the reagent pulse was injected through the reactor and trapped also. On heating the trap all the material collected there passed into the GC. If correct identification had been made, an enlargement of the relevant peak was observed due to the extra material added to the products of reaction. Incorrect identification, however, resulted either in another peak appearing or in the enlargement of a different product peak.

Conclusion

In summary, the design, assembly and testing of the apparatus and techniques necessary to the project proceeded well except that, in general, all stages of the project required longer to complete than planned and the ECD failed to meet appropriate levels of reproducibility and reliability. The increased time necessary for each stage of the preliminary work was due to the extra efforts made to make the ECD fit for use.

It was decided to use the furnace in the manually controlled pulse firing mode, rather than the process control mode as there was a considerable lag time between the coil heating and an increase in furnace temperature. Satisfactory control and stability of the reactor temperature was possible using this method of control.

The replacement of the gas system with stainless steel tubing was completed and the system appeared leak free and, therefore, dry, within the detection limits available. However, later work (see Chapter 5) indicates that the system was not dry. It is possible that this is related to the problem in making the ECD reliable enough to use.

It was decided that the ECD was too unreliable to use and that the FID, given its unusually good performance, would be the only detector used for this work. Although this was not the original plan, which was to use the complementary qualities of the two detectors, no other option was possible.

Chapter 4 : Investigation of the Pyrolysis of 1,1,1-trichloroethane

Introduction

The aim of this work was to determine the efficiency of the batch stirred-flow technique for investigating the pyrolyses of organochlorine compounds whilst pursuing such an investigation for 1,1,1-trichloroethane (TCE). The pyrolysis of TCE was studied because of its similarity to the central moiety of the DDT molecule which should allow it to serve as a model for the decomposition of this more complicated material. Also, the pyrolysis of TCE in the temperature range 623 - 723 K has the advantage of having received much attention in the literature,[45-52] (comparison of experimental results for TCE pyrolysis with the literature should allow a judgement of the reliability of the techniques and apparatus. The basis of such comparisons are the key parameters of the reaction, e.g. activation energy, Arrhenius pre-exponential factor, with the literature values).

The broad features of the mechanism of TCE pyrolysis were first detailed by Barton *et al*[45] in 1950. Barton *et al*[45] found that in clean-walled glass vessels, in the temperature range 635 - 707 K, the reaction was fast and, at least partly, heterogeneous. After a time, a carbonaceous layer built up on the inside of the reactor, the rate gradually decreased and, eventually, became reproducible, (it is usually assumed by workers in the field that, under these conditions, the reaction can be considered to be essentially homogeneous,[46] i.e. most of the heterogeneous reaction is suppressed by the carbonaceous layer on the reactor walls).

The major reaction taking place in the temperature range 635 - 707 K was the dehydrochlorination of TCE to give hydrogen chloride and 1,1-dichloroethene (Scheme 7). (The possibility of further pyrolysis of 1,1-dichloroethene has been investigated[45] but was found to be negligible in this temperature range). The TCE dehydrochlorination followed first order kinetics until ~40% conversion at initial 1,1,1-trichloroethane pressures of 9.3-10.7 kPa. At TCE partial pressures below 9.3 kPa the rate constant showed a marked decrease. Above this pressure, i.e. in the range 9.3-10.7 kPa and, indeed, up to 13.3 kPa, the rate constant became insensitive to initial pressure of TCE. The deviation of the kinetics from

linear at greater than 40% conversion was thought to derive from inhibition of the reaction by one of the products and, indeed, 1,1-dichloroethene was found to inhibit the reaction by an amount roughly proportional to its concentration.

$$CH_3CCl_3 \longrightarrow HCl + CH_2{=}CCl_2$$

Scheme 7 Dehydrochlorination of 1,1,1-trichloroethane (TCE) to give hydrogen chloride and 1,1-dichloroethene

Barton *et al*[45] found the pyrolysis to be slightly dependent on surface area despite the assumption, reported above, that the reaction was homogeneous under the conditions used. Two reactors were used to study the effect of surface area, one empty and one packed with hollow glass tubes. Increased surface area caused a decrease in both the rate and the induction period. However, the induction periods in both the packed and empty reactors had definite temperature dependencies, given by :

$$\text{I (empty reactor)} = 10^{-14.4} \cdot e^{50000/RT} \text{ sec}$$
$$\text{I (packed reactor)} = 10^{-13.4} \cdot e^{50000/RT} \text{ sec}$$

The decrease in rate and induction period can be explained by the existence of an accelerating effect of the surface on the termination and initiation reactions for the radical process. The initiation reactions, being faster, would decrease the time needed to reach a viable concentration of reactive species, i.e. the induction time, and increase the rate. Faster termination steps, however, would tend to slow the overall rate. (In the present case, given such an explanation, the evidence points to the effect on the termination steps being more important than that on the initiation steps, since the rate decreases with increased surface area, not vice versa). Thus the only observable effect of the initiation acceleration is to decrease induction times. However, while all the above is consistent with the results reported by Barton *et al*,[45] the only firm conclusion is that the reaction contains a variety of components.

Table 2 Arrhenius parameters for the pyrolysis of 1,1,1-trichloroethane, or various components of that reaction

		$\log_{10}$ (A / s)[a]	E_a / kJ mol^{-1}	Ref. #
Overall reaction	Empty reactor	13.34 ± 0.27	210 ± 4.2	45
	Packed reactor	13.21 ± 0.26	210 ± 4.2	45
Maximally inhibited reaction (i.e. molecular component)	Empty reactor	14.07 ± 0.28	227.6 ± 4.5	45
	Packed reactor	13.95 ± 0.28	225.3 ± 4.5	45
		13.85 ± 0.14	226.8 ± 2.3	52

Molecular Component of Pyrolysis

All the features of the maximally inhibited reaction, mentioned above, indicate that it is molecular in character. Propene is known to inhibit many radical chain processes by severely decreasing the concentration of radical chain carriers and, therefore, is used to test for the presence of such processes. Equally, the fact that the maximally inhibited pyrolysis does not have an induction period (a characteristic of radical chain reactions) provides further evidence for the molecular character of the reaction. Finally, at TCE partial pressures greater than 9.3 kPa, the independence of the rate on initial pressure of TCE provides another reason to believe the inhibited reaction to have a molecular mechanism as this phenomenon is due to competing radical chain termination steps (see page 56). Barton *et al*[45] support this argument further by quoting from a literature survey a usual value for the pre-exponential factor (A) for a unimolecular gas-phase reaction of 10^{13} and pointing out that the experimental value is within an order of magnitude of this.

---

[a] Errors are one standard deviation from least squares regression analysis of data

Radical Chain Component of Pyrolysis

Barton *et al*[45] concluded that the major portion of the pyrolysis (over 60% under the conditions used) has a radical chain mechanism. Their evidence was that :

(a) the overall reaction is suppressed by small amounts of propene (a well known inhibitor of radical chain reactions),

(b) the reaction is generally slower in the packed reactor and

(c) the presence of induction periods with a regular dependence on temperature.

Point (b) above (the reaction is slower in the packed reactor) is slightly ambiguous as molecular reactions also can be effected by surfaces. However, while it is true that acceleration due to large surface area could be due to catalysis either of a molecular reaction or of a radical chain mechanism, it is inhibition that is observed rather than acceleration. It is likely that a radical chain termination step is catalysed at the surface and that this decreases the chain length of the radical reaction and, hence, the rate. Further evidence for this conclusion is provided by the pressure dependence of the rate constant, which decreases with initial pressure indicating two termination steps, one heterogeneous and one homogeneous. As pressure decreases, within the range 9.3-13.3 kPa, the homogeneous reaction becomes less important, depending as it does on the amount of reactant in the gas phase. The heterogeneous step becomes dominant as diffusion to the wall becomes more rapid, with less chance of homogeneous encounters.

Based on the evidence given above, Barton *et al*[45] proposed a reaction mechanism for the decomposition of TCE :

Initiation

$$CH_3CCl_3 \xrightarrow{k_{1a}} {}^{\bullet}CCl_2CH_3 + {}^{\bullet}Cl \quad (a)$$

$$CH_3CCl_3 \xrightarrow{k_{1b}} {}^{\bullet}CH_3 + {}^{\bullet}CCl_3 \quad (b)$$

Propagation

$${}^{\bullet}Cl + CH_3CCl_3 \xrightarrow{k_2} HCl + {}^{\bullet}CH_2CCl_3 \quad (c)$$

$${}^{\bullet}CH_2CCl_3 \xrightarrow{k_3} CH_2{=}CCl_2 + {}^{\bullet}Cl \quad (d)$$

Termination

$${}^{\bullet}CH_2CCl_3 + {}^{\bullet}Cl\ (\text{wall}) \xrightarrow{k_4} \text{Products} \quad (e)$$

$${}^{\bullet}CH_2CCl_3 + {}^{\bullet}Cl\ (\text{homogeneously}) \xrightarrow{k_5} \text{Products} \quad (f)$$

Scheme 8 Reaction Mechanism Proposed by Barton *et al*[45] for the Pyrolysis of TCE

The effects of steps (a) and (b) were held to be indistinguishable by Barton *et al*,[45] if in the case of (b) the additional steps

$${}^{\bullet}CH_3 + CH_3CCl_3 \longrightarrow CH_4 + {}^{\bullet}CH_2CCl_3 \quad (g)$$

$${}^{\bullet}CCl_3 + CH_3CCl_3 \longrightarrow CHCl_3 + {}^{\bullet}CH_2CCl_3 \quad (h)$$

$${}^{\bullet}CH_2CCl_3 \longrightarrow CH_2{=}CCl_2 + {}^{\bullet}Cl \quad (i)$$

were included, since it was not possible at that time to calculate differences in rate sufficiently accurately to use them in a predictive manner. However, it was assumed that only one initiation step would be important under the conditions used. They went on to use the steady state approximation to quantify the theory to test whether the competition between heterogeneous and homogeneous termination reactions was a valid mechanism. Using reactions (a), (c), (d) and versions (e) and (f), the following expression for the rate of the radical chain process was derived:

$$\frac{-d[CH_3CCl_3]}{dt} = [CH_3CCl_3]\left(\frac{k_{1(a\ or\ b)}k_2k_3}{2(k_4+k_5)}\right)^{1/2} \qquad \textbf{Equation 19}$$

Barton *et al*[45] assume that a normal treatment of heterogeneous chain reactions is valid, i.e. that the rate limiting feature of the heterogeneous termination step is diffusion of the species to the wall.[47a] Therefore, as the velocity constant (k_4) depends upon the rate of diffusion and the diffusion rate depends upon the square of the pressure in the container, the diameter of the container and the diffusion coefficient, k_4 is given by :

$$k_4 = \frac{k_4'}{p_0^2} \qquad \textbf{Equation 20}$$

where :

k_4' is a value, independent of pressure, composited from the diameter of the vessel and the diffusion coefficient

p_0 is the initial pressure of reactant in the reactor, taken as representing the overall pressure for the initial part of reaction

On replacing this in the overall rate equation :

$$k_{overall} = k_i + k_s \qquad \textbf{Equation 21}$$

where :

k_i rate constant for the maximally inhibited reaction

k_s rate constant for the suppressible (i.e. radical chain) part of the reaction)

One obtains :

$$k_{overall} = k_i + \frac{\left(\frac{k_1 k_2 k_3}{2k_4^{'}}\right)^{1/2}}{\left(\frac{1}{p_0^2} + \frac{k_4^{''}}{k_4^{'}}\right)^{1/2}}$$

$$= k_i + \frac{C_1}{\left(\frac{1}{p_0^2} + C_2\right)^{1/2}} \qquad \textbf{Equation 22}$$

As C_1 and C_2 are both combinations of *homogeneous* rate constants, the ratio :

$$\frac{C_1^2}{C_2} = \frac{k_1 k_2 k_3}{2k_4^{''}} \qquad \textbf{Equation 23}$$

should be independent of the reaction vessel, i.e. it should be the same for the empty and packed reactor. On testing this, Barton *et al* found that it was indeed independent of reaction vessel, thereby supporting the assumption of homogeneous vs. heterogeneous termination step competition.

In summary, Barton *et al*[45] established that the only major reaction taking place in the pyrolysis of 1,1,1-trichloroethane is dehydrochlorination to 1,1-dichloroethene. The reaction proceeds by two competing mechanisms, one a radical chain process, the other molecular. The features of the radical chain process are that it is inhibited by 1,1-dichloroethene (a product) and very small additions of propene. The reaction proceeds more slowly in a reaction vessel with high surface area and the rate constant has a complicated pressure dependence. There were also definite induction periods which were dependent on the temperature. The molecular mechanism, however, is independent of pressure and surface area and does not have induction periods.

Much of the more recent work on this subject has been concerned with refining the details of the mechanism. The pyrolysis of 1,1,1-trichloroethane by use of a CO_2 laser has been reported by Pola *et al*[48]. As the laser beam diameter is smaller than the diameter of the cylindrical reaction vessel the heating should not effect the gas near the walls. For this reason, that heterogeneous processes are assumed to be negligible, only homogeneous processes need be considered.

Pola[48] highlights two views concerning the inhibitive effect of propene on the dehydrochlorination of TCE and tries to distinguish between them using his results. The first theory, which Pola[48] attributes to Swinbourne,[49] is that propene acts by converting normal radical-chain carriers into less efficient species, thereby retarding the reaction. Pola rather inconsistently argues that the laser-induced pyrolysis of 1,1,1-trichloroethane is a non-radical, molecular decomposition. From this argument, one would suppose that no retarding effect by propene on the laser-induced dehydrochlorination of TCE had been observed since a molecular reaction would have no chain carriers to be effected by propene. In fact, there was a 50% decrease in rate on propene addition. Therefore, it does not seem to be consistent that Swinbourne's explanation[49] of the propene effect implies a molecular reaction for the laser-induced pyrolysis observed by Pola.[48]

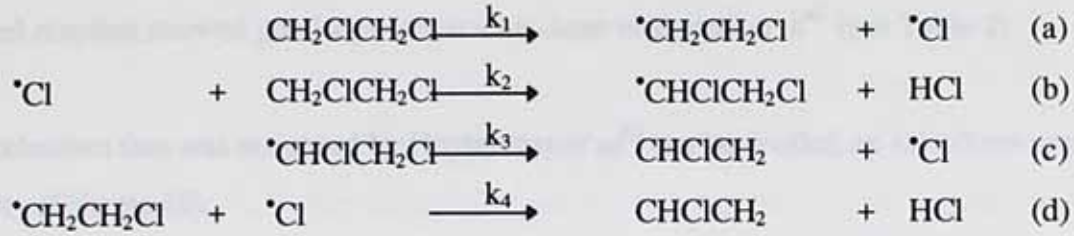
The second explanation of propene inhibition that Pola[48] considered (attributed to Zitter *et al*[50]) assumes that the radical chain pyrolysis is initiated at the walls of the reaction vessel. Propene would have the effect of deactivating sites on the walls thus inhibiting the initiation of the chain reaction. One of the fundamental assumptions made in interpreting Pola's laser pyrolysis data, however, is that no heterogeneous processes are involved in the reaction. Therefore any process taking place here must be homogeneous and Pola's attempt to use Zitter's explanation[50] of the propene effect must be viewed as extremely suspect. However, Pola does present a convincing argument, based on an abstraction competition, that the pyrolysis under these conditions is molecular in character. One could speculate that this is because of the limitation placed on initiation steps due to the precise delivery of energy by the laser. While providing little enlightenment on the mechanism by which propene inhibits the radical chain reaction, as Pola apparently intended, it does open up the possibility of studying the molecular reaction directly, i.e. in the absence of propene additions, the usual method of eliminating radical reaction.

The Rice-Herzfeld mechanism and its validity for halogenated hydrocarbon pyrolyses

Huybrechts *et al*[51] investigated the pyrolysis of 1,2-dichloroethane and found that, in common with several other compounds (1,1,2-trichloroethane, 1,1,2,2-tetrachloroethane and pentachloroethane), values of Arrhenius parameters calculated from theory using a Rice-Herzfeld[47b] mechanism (Scheme 9) were significantly different from the experimental values. At 700 K the values for the rate constants from experiment and theory differ by more than three orders of magnitude.

Theoretical	$\log_{10}(k / s) = -(50\,900 \pm 2\,000) / 4.58\ T + (\leq 15.6 \pm 0.7)$
Experimental	$\log_{10}(k / s) = -47\,000 / 4.58\ T + 10.8$

Due to the poor agreement between theory and experiment, Huybrechts *et al*[51] suggest that the Rice-Herzfeld mechanism itself must be questioned in its applicability to this type of reaction. However, part of the calculation to determine the theoretical value assumes the ratio k_1/k_4 to be limited by the equilibrium (a)/(-a), for which the authors do not seem to have given any evidence. The uncertainty over the calculation made on the basis of the above assumption brings the entire argument into doubt, leaving the question unresolved. It might be more useful to question instead, not the basic validity of the Rice-Herzfeld mechanism but its simplicity. As will be discussed below, additions to the Rice-Herzfeld mechanism are often necessary in order to explain all the experimental data for a particular organohalogen pyrolysis. This model was developed originally to explain hydrocarbon pyrolyses (for which it works very well) so it would not be surprising if changes were necessary to incorporate the more complex halogenoalkane case.



Scheme 9 Rice-Herzfeld type mechanism for the pyrolysis of 1,2-dichloroethane[51]

The Effect of Additives and a Modified Mechanism

Huybrechts *et al*[52] investigated the TCE pyrolysis in the presence and absence of tetrachloromethane and hydrogen chloride, both of which were found to accelerate the reaction. The temperature range of study was 587 - 658 K, with pressure ranging between 5.33 and 20.2 kPa, i.e. similar to the conditions used by Barton *et al.*[45] Essentially the same products of reaction (hydrogen chloride and 1,1-dichloroethene [CH_2CCl_2]) were found in both studies although there were several other, trace compounds (<0.05 %) detected also. Results were held to apply to the homogeneous process as there was no significant change in the results with changing surface to volume ratio. Again, a combination of a radical chain reaction and a molecular decomposition were posited to explain the inhibitive effect of propene. The reaction is auto-accelerated and dependent on initial pressure of 1,1,1-trichloroethane in a complicated way.

The auto-acceleration was investigated by additions of hydrogen chloride, the effect becoming less marked as larger hydrogen chloride additions were made, indicating that hydrogen chloride production is responsible for the auto-acceleration in the unmodified reaction. In the presence of added tetrachloromethane and mixtures of tetrachloromethane and hydrogen chloride, the dehydrochlorination is still the only reaction of significance. The rate is accelerated in the presence of tetrachloromethane alone and this acceleration is independent of the amount added. Acceleration due to hydrogen chloride, however, depends on the concentration. It is obvious from this that the mechanisms by which the accelerations due to tetrachloromethane and hydrogen chloride occur are significantly different. Again, propene was demonstrated to

have a strong inhibitive effect on the reaction and Arrhenius parameters for the maximally inhibited reaction showed good agreement with those of Barton *et al*[45] (see Table 2).

The mechanism that was proposed by Huybrechts *et al*[52] can be divided up into three groups of reaction (Scheme 10):

a) the Rice-Herzfeld mechanism (a group of reactions often used to explain the pyrolyses of hydrocarbon compounds[47b])

b) a group of transfer reactions that change δ radicals (radicals that do not perpetuate the chain reaction) into chain carrying μ and β radicals (radicals that do perpetuate the chain propagation, in primary and secondary reactions respectively),

c) a group consisting of reactions that take place in the presence of tetrachloromethane.

The reactions of the last group, whilst not strictly relevant to the pyrolysis of pure TCE, nevertheless do help to elucidate important parts of the mechanism.

The point at which the Rice-Herzfeld mechanism becomes insufficient to explain the 1,1,1-trichloroethane pyrolysis is in the auto-catalytic effect of hydrogen chloride. That such an effect exists is shown quite clearly by the results reported in the currently discussed paper. Also, an autocatalytic effect had been observed by Barton *et al*[45] in the first investigation of this reaction. However, as Huybrechts *et al*[52] point out, the possibility that hydrogen chloride catalysis could have been taking place was not considered and it was assumed that a slow build up of radicals was the cause. Huybrechts *et al*[52] dismissed this explanation in view of the fact that in simulation a steady-state was reached in approximately eighty seconds which is two orders of magnitude less than the time-scale over which the catalytic effect exists. In the simulations the catalytic effect manifests as a great change in the concentration-time profiles of the radicals. The reaction without additional hydrogen chloride shows a large build up of non-chain carrying radicals (δ) early in the reaction which rapidly decreases in the latter stages of reaction (due to conversion of δ radicals into chain carrying μ radicals (radicals reacting in first order steps) and β radicals (radicals taking part in second order steps). In the presence of additional hydrogen chloride, however, there is a rapid rise in concentration of μ and β radicals after which a slow increase is maintained. The δ radicals do not reach such a high level. Indeed, it is the fact that the presence of hydrogen chloride allows fast conversion of δ radicals

(species marking the end of chain sequences) to chain carriers that is responsible for much of the catalytic effect.

The presence of tetrachloromethane accelerates the reaction in two ways. Firstly, there is the production of Cl radicals, a chain carrying species, and, secondly, extra transfer reactions. Huybrechts *et al*[52] report that this part of the simulation was very successful in its reproduction of the experimental observation that the degree of acceleration did not depend on the amount of tetrachloromethane added.

In summary, it can be said that the pyrolysis of TCE has been characterised well by the workers of the last 45 years. The high temperature (>1000 K) pyrolysis has received some attention also,[53, 54] but not to the same extent as the lower temperature reaction. The literature has allowed a fair degree of certainty in making predictions about the probable findings of this investigation. However, this work was undertaken in part to determine whether these predictions were met. The comparison between literature and experiment was one of the planned testing stages for the batch stirred-flow reactor method used in this work.

Rice-Herzfeld mechanism			CCl_3CH_3	$\xrightarrow{k_1}$	$^{\bullet}CCl_2CH_3$	+	$^{\bullet}Cl$
	$^{\bullet}Cl$	+	CCl_3CH_3	$\xrightarrow{k_2}$	$^{\bullet}CH_2CCl_3$	+	HCl
			$CCl_3CH_2^{\bullet}$	$\xrightarrow{k_3}$	$CCl_2=CH_2$	+	$^{\bullet}Cl$
	$^{\bullet}Cl$	+	$^{\bullet}CCl_2CH_3$	$\xrightarrow{k_{-1}}$	CH_3CCl_3		
Transfer Reactions	HCl	+	$^{\bullet}CCl_2CH_3$	$\xrightarrow{k_5}$	CCl_2HCH_3	+	$^{\bullet}Cl$
	$^{\bullet}CCl_2CH_3$	+	CCl_3CH_3	$\xrightarrow{k_6}$	CCl_2HCH_3	+	$CCl_3CH_2^{\bullet}$
	$^{\bullet}Cl$	+	CCl_3CH_3	$\xrightarrow{k_7}$	$^{\bullet}CCl_2CH_3$	+	Cl_2
Reactions involving CCl_4			CCl_4	$\xrightarrow{k_8}$	$^{\bullet}CCl_3$	+	$^{\bullet}Cl$
	$^{\bullet}CCl_3$	+	HCl	$\xrightarrow{k_9}$	CCl_3H	+	$^{\bullet}Cl$
	$^{\bullet}CCl_3$	+	CCl_3CH_3	$\xrightarrow{k_{10}}$	CCl_3H	+	$CCl_3CH_2^{\bullet}$
	CCl_4	+	$^{\bullet}Cl$	$\xrightarrow{k_{11}}$	$^{\bullet}CCl_3$	+	Cl_2
	$^{\bullet}CCl_3$	+	$^{\bullet}Cl$	$\xrightarrow{k_{-8}}$	CCl_4		
	$^{\bullet}CCl_3$	+	$^{\bullet}CCl_3$	$\xrightarrow{k_{12}}$	CCl_3CCl_3		

Scheme 10 Total mechanism proposed by Huybrechts *et al*[52]

Experimental

$$CCl_4 \xrightarrow{k_8} {}^{\bullet}CCl_3 + {}^{\bullet}Cl$$

$${}^{\bullet}CCl_3 + HCl \xrightarrow{k_9} CCl_3H + {}^{\bullet}Cl$$

$${}^{\bullet}CCl_3 + CCl_3CH_3 \xrightarrow{k_{10}} CCl_3H + CCl_3CH_2^{\bullet}$$

$$CCl_4 + {}^{\bullet}Cl \xrightarrow{k_{11}} CCl_4 + Cl_2$$

$${}^{\bullet}CCl_3 + {}^{\bullet}Cl \xrightarrow{k_8} CCl_4$$

$${}^{\bullet}CCl_3 + {}^{\bullet}CCl_3 \xrightarrow{k_{12}} CCl_3CCl_3$$

All kinetic experiments were carried out in a greaseless reaction apparatus comprising stainless steel gas-supply lines, a pyrex or quartz glass reactor and gas-sampling valve internally coated with teflon (see Chapter 3 for more details), with helium as the carrier gas in the temperature range 470 - 770 K. All chromatograms were run using a Perkin-Elmer 8500 machine with electronic integrator and a flame ionisation detector (FID). 1,1,1-trichloroethane and 1,1-dichloroethene were degassed, firstly by successive freeze, pump and thaw cycles and secondly by trap to trap distillation under vacuum. Products were identified by a combination of retention time comparison and by addition of an authentic sample of the material to the pyrolysis mixture after reaction, giving an enlarged peak which was taken as positive identification. Reactant pressures were measured before injection into the reaction system by a baratron linked to a Chell Ltd digital readout.

Results and Discussion

Only one major product of pyrolysis was detected under the reaction conditions used. This compound was identified as 1,1-dichloroethene (CH_2CCl_2) using gas chromatography (GC), both by retention time comparison with an authentic sample of the material and by standard additions of this chemical to the product mixture before analysis. This result is consistent with the literature,[45,48] which also reports hydrogen chloride as a major product of 1,1,1-trichloroethane pyrolysis. However, as the detector used here was a Flame Ionisation Detector (FID), which operates by detecting an ionic species resulting from the combustion of organics, it was not expected that hydrogen chloride would be observed. Once the identity of the observable product had been established it was decided to measure the order of reaction with respect to the product in 1,1,1-trichloroethane.

Order of Reaction for 1,1,1-trichloroethane with respect to. 1,1-dichloroethene

In order to measure the order of reaction using the batch stirred-flow technique the amounts of reactant (1,1,1-trichloroethane) and product (1,1-dichloroethene) leaving the reactor need to be known for a series of experiments starting with different amounts of reactant. The amounts of materials leaving the reactor were measured by a gas chromatograph previously calibrated with pure samples of the compounds. Having obtained the amounts of material exiting the reactor, a plot of log(product) against log(reactant) gives a line of slope n (order of reaction) as[42,43] :

$$\log(\text{product}) = n \cdot \log(\text{reactant}) + \log\left\{\frac{k\tau}{v^{(n-1)}n}\right\} \qquad \textbf{Equation 24}$$

where :

k = rate constant v = volume of the reactor

τ = time constant n = order of reaction

Although it was possible to obtain data that fit the above equation well (Figure 17), the results of the order experiments were not reproducible over the temperature range used and, on analysis, gave values for the order of :

pyrex reactor (volume = 18.6 cm^3)	0.86 ± 0.11
quartz reactor (volume = 22.4 cm^3)	0.84 ± 0.37

Although the value for the quartz reactor is in agreement with the literature value[45] of 1.0, within experimental error, the error given (one standard deviation) is too large to allow the data to be treated with any trust. Whilst it might be tempting, on obtaining results that are not truly in agreement with the literature, to assume that a new and interesting chemical problem was being observed, the irreproducibility of the data led us to believe it more probable that some unknown and uncontrolled factor was interfering with the pyrolysis.

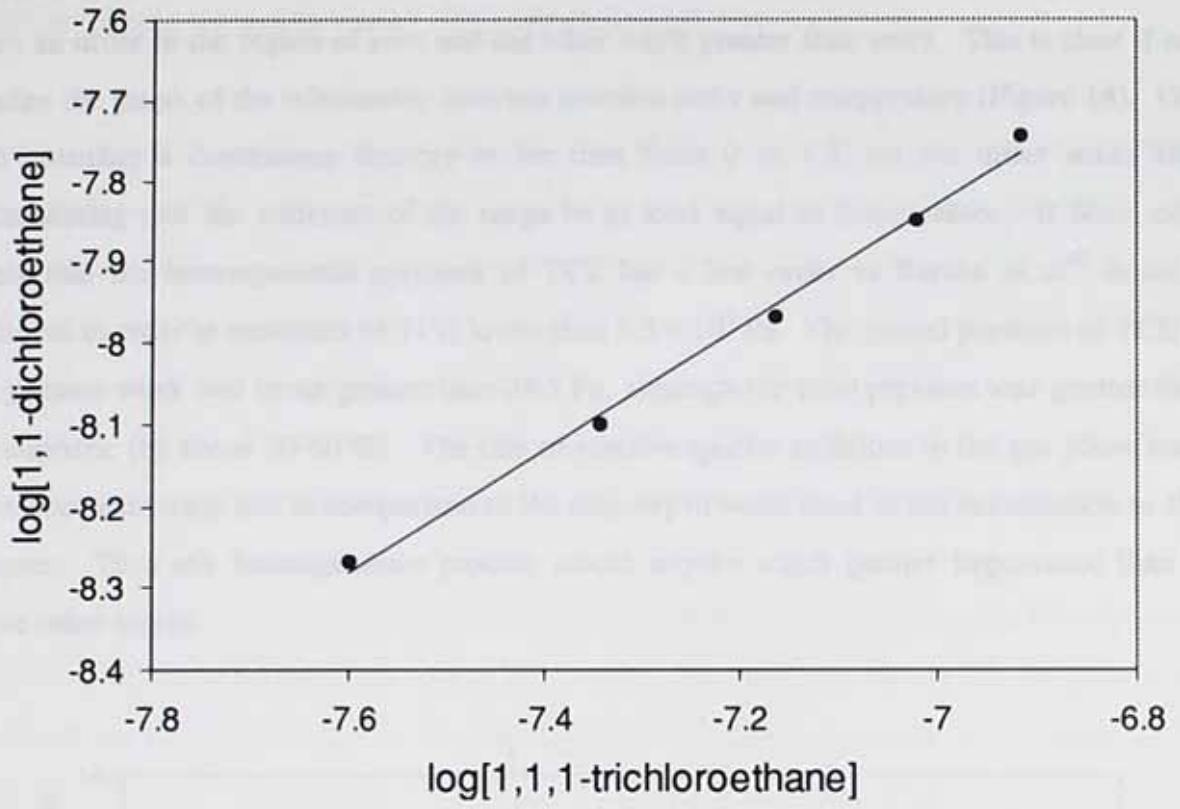


Figure 17 Order plot for the pyrolysis of 1,1,1-trichloroethane in the pyrex reactor at 691 K, τ = 55.2 s.

A possible candidate for such a factor became apparent when it was noticed that there seemed to be some trend in the value of the reaction order with temperature. On extending the temperature range and number of order determining experiments it was possible to see a definite relationship between the two parameters (Figure 18). The linear regression line shown is given by the equation :

$$n = 0.0093 \cdot T - 4.372$$

where n is the order of reaction and T is the temperature. The correlation coefficient, R, was 0.91. Although this does not represent a perfect correlation (which corresponds to a value of unity), it certainly indicates a real effect.

It is possible that this result indicates the presence of two competing mechanisms with different orders and activation energies. Likely candidates for such a competition are a homogeneous and a heterogeneous decomposition.[45,52] Such a phenomenon is not unknown in this type of batch stirred-flow reactor and would fit with TCE's known tendency for heterogeneous

decomposition.[45,48,51,52] For such a model it would be necessary for one of the processes to have an order in the region of zero and the other much greater than unity. This is clear if one studies the graph of the relationship between reaction order and temperature (Figure 18). One can postulate a continuous linearity in the data from 0 to 1.5, on the order scale, thus necessitating that the extremes of the range be at least equal to these values. It does seem likely that the heterogeneous pyrolysis of TCE has a low order as Barton *et al*[45] found a decrease in order at pressures of TCE lower than 9.3×10^3 Pa. The partial pressure of TCE in the present work was never greater than 24.5 Pa, although the total pressure was greater than atmospheric (by about 50-60 %). The rate of reactive species collisions in the gas phase must have been extremely low in comparison to the majority of work cited in the introduction to this chapter. Thus any heterogeneous process would acquire much greater importance than in those other works.

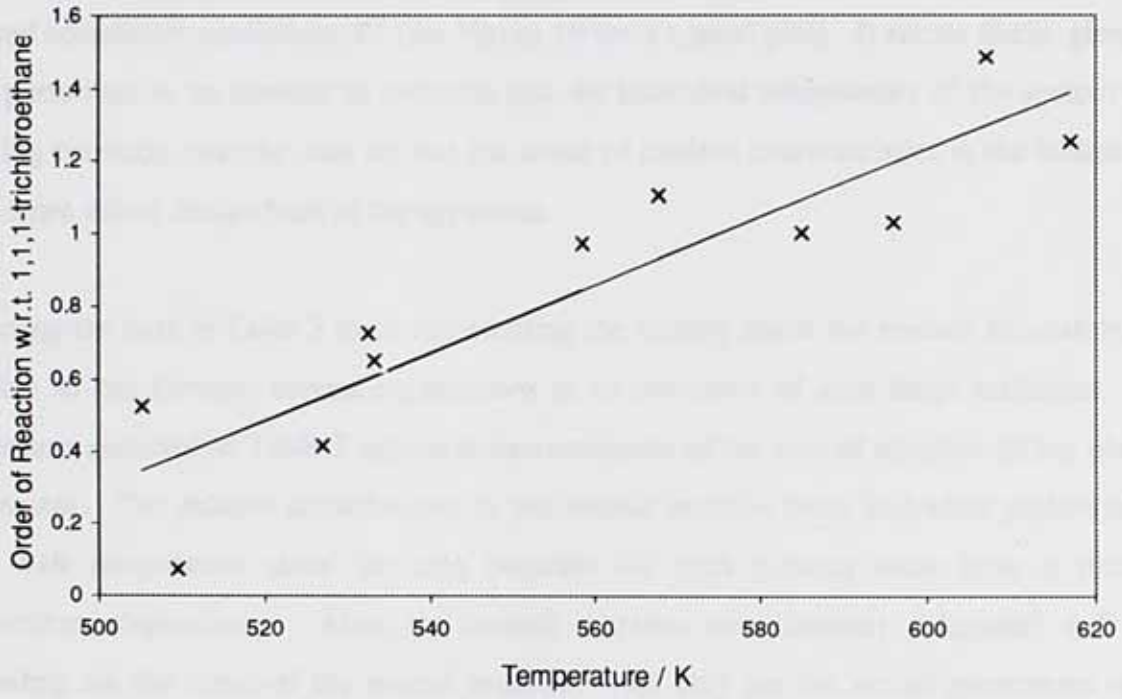


Figure 18 **Graph showing the relationship between reaction order and temperature for the pyrolysis of 1,1,1-trichloroethane in a quartz reactor**

Temperature dependence of the rate constant

It should be noted that the accurate determination of the rate constant using the batch stirred-flow reactor method requires the determination of the order of reaction to be reliable[42,43] (see Pulse Stirred-Flow in Chapter 2). Determinations of the rate constant are very sensitive to variation in the order of reaction, as well as to other factors normally associated with such measurements, e.g. pressure and surface effects. Since, as shown above, the experimentally determined order of reaction appeared to change with temperature, the temperature dependence of the rate constant would be expected to be more than normally complicated.

The variation in the results shown in Table 3 indicate either that the apparatus is behaving very badly or that the pyrolysis occurring is not simple. Not only are the Arrhenius parameters extremely varied, so is the degree with which the Arrhenius plots are linear, as measured by the squared correlation coefficient, R^2 (see Figure 19 for a typical plot). It seems likely, given the tests performed in an attempt to ascertain that the individual components of the system were working correctly, that the data are not the result of random characteristics in the instruments but a more subtle design fault of the apparatus.

Assuming the data in Table 3 to be representing the activity inside the reactor accurately, it is possible to put forward several explanations as to the cause of such large variations. The parameters reported in Table 3 represent measurements of the overall reaction taking place in the reactor. The relative contributions to the overall reaction from individual pathways will vary with temperature since the rate constant for each process must have a different temperature dependence. Also, the overall reaction rate constant calculated will vary depending on the order of the overall reaction. Not only are the actual parameters of the overall process changing as the magnitude of the individual components change, these variations are exaggerated by the corresponding change in measured reaction order which has been used to calculate those parameters.

Table 3 **Showing Arrhenius parameters of several investigations of the pyrolysis of 1,1,1-trichloroethane**

Activation Energy / kJ mol^{-1}	Log A	Square of Correlation Coefficient (R^2)
198.7	18.1	0.921
148.0	11.9	0.991
95.0	6.6	0.937
108.9	8.0	0.987
166.3	12.6	0.987
168.9	11.9	0.986
71.8	3.6	0.936

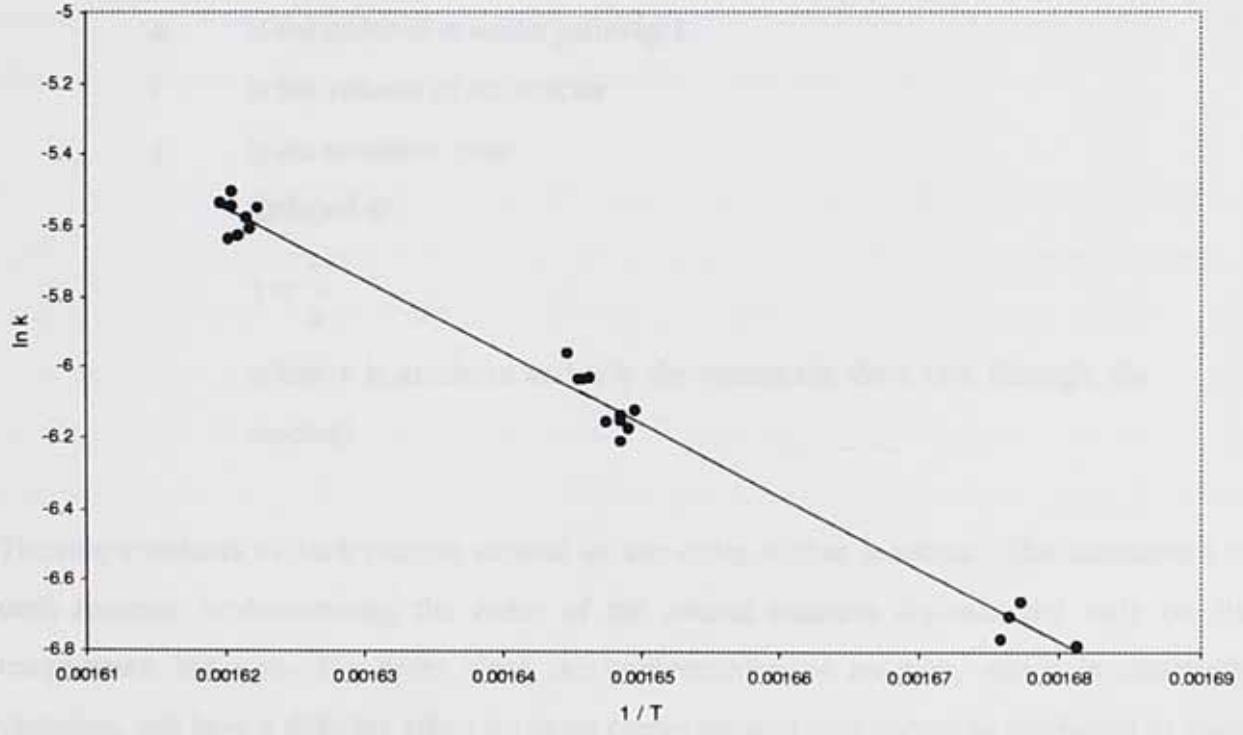


Figure 19 **Arrhenius plot for the pyrolysis of 1,1,1-trichloroethane**

Attempts to define mathematically the dependence of all the parameters on temperature and thus, perhaps, allow characterisation of the individual components of the reaction have not

been successful. The complexity of the problem can be seen if the overall rate constant is defined as :

$$k = k_1 + k_2 + \ldots\ldots, k_i \qquad \textbf{Equation 25}$$

where k_1, k_2, etc. are the rate constants of all the separate, competing pathways in the dehydrochlorination. Each rate constant is defined as[42] :

$$k_i = \frac{(B_i)v^{(n_i-1)}n_i}{(A_o)^{n_i}\tau} \qquad \textbf{Equation 26}$$

where k_i is the rate constant for reaction pathway i

(B_i) is the amount of B collected at the outlet due to reaction pathway i

(A_o) is the amount of A collected at the outlet

n_i is the order of reaction pathway I

v is the volume of the reactor

τ is the residence time

(defined as :

$$\tau = \frac{v}{u}$$

where v is as above and u is the volumetric flow rate through the reactor)

The rate constants of each process depend on the order of that reaction. The importance of each reaction in determining the order of the overall reaction depends not only on the temperature but upon the order since the concentration of reactant, which is constantly changing, will have a differing effect on those processes with high orders as compared to those with low orders. In the light of the complexity of the inter-relationship of the various Arrhenius parameters it is not surprising that the results obtained seem random. The system under study has too many uncontrolled degrees of freedom to allow proper analysis.

It is of interest to consider what the competing processes might be. As stated above, the most likely case is a competition between homogeneous and heterogeneous dehydrochlorination routes. The extremely small size of the injections (of the order of 10^{-8} moles or 4×10^{-6} g) of reactant, 1,1,1-trichloroethane, makes them inadequate for the conditioning of the reactor which normally takes place in conventional kinetic investigations of organochlorine pyrolyses.[45,46,51,52] It is ironic that this design characteristic is a consequence of one of the biggest advantages of this system, that is, the necessity to use only very small amounts of compound. In order to test this hypothesis it would be necessary to use a reactor that was known to be incapable in participating in organochlorine pyrolyses. There are several ways of making glass surfaces less reactive but, according to the best information available, the highest temperature technique is useless above *ca.* 523-573 K. The alternative is to use different materials for the reactor but it is hard to find one that is suitable.

Further problems may have been caused if the reaction system had a leak, allowing an influx of water vapour and/or oxygen from the atmosphere. It seems likely, given the radical chain nature of the pyrolysis, that small amounts of additives could have a disproportionately large effect on the mechanism, in a similar manner to that of propene.

In an attempt to test the above hypotheses regarding problem(s) with the reaction system, a new investigation was undertaken into the pyrolysis of 1,1-dimethyl-1-silacyclobutane, a compound on which much work has been performed and which was used by the Davidson group to test the batch stirred-flow technique during its development (see Chapter 5 for this investigation). The results of the investigation showed that there was water present in the reaction system. If the water came from an atmospheric leak, as seems likely given the purity of the gas supplies used, then the reaction system might have contained significant levels of oxygen also.

Conclusion

The values obtained from the investigation of the pyrolysis for the Arrhenius parameters and the order of reaction exhibited extreme variations in every way. It is possible to postulate two types of explanation for this. Either the apparatus is faulty in some way such that the values obtained for parameters of interest are not related to the actual processes occurring, or the values represent a reasonably accurate picture of the chemistry taking place and some other factors are causing this to be other than expected. It seems more likely that the apparatus is giving a reasonable approximation of the system and that the variation in the observed Arrhenius parameters is due to changes in a complex system of separate, competing decomposition pathways as a result of the water and, perhaps, oxygen present in the apparatus and of interactions with reactive vessel walls. The presence of water in the system was demonstrated through the study of the pyrolysis of a silanyl compound, 1,1-dimethyl-1-silacyclobutane. Unfortunately, this information came at too late a stage in the investigation to allow a solution for the problem to be found.

Chapter 5 : Investigation of the Pyrolysis of 1,1-Dimethyl-1-silacyclobutane

Introduction

The work in this chapter was undertaken in an attempt to solve the problems with the 1,1,1-trichloroethane (TCE) pyrolysis. The intention was to ascertain the state of the reaction apparatus during a decomposition experiment by comparing the results of studies on the title compound's decomposition with accounts given in the literature. The first group to develop the batch stirred-flow methodology for kinetic work[43] used the pyrolysis of 1,1-dimethyl-1-silacyclobutane as a test reaction for their apparatus. It was chosen for reasons of simplicity and because there is a definite change in reaction products and rate in the presence of water[55] (see below).

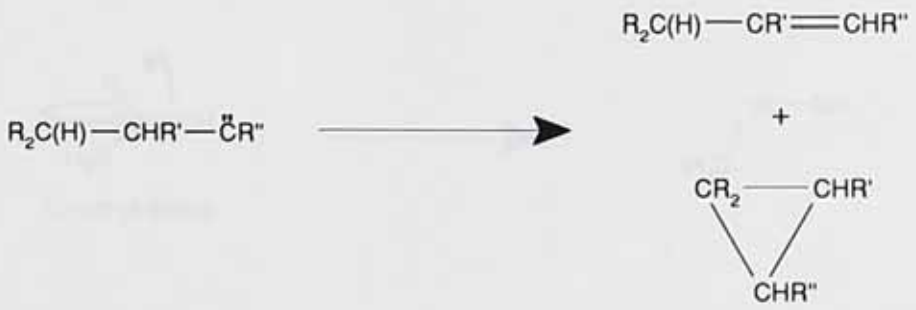
Silylenes

Silylenes were first prepared and observed spectroscopically in 1937 by Schwartz *et al*[56] but were not investigated in any detail for approximately twenty years. Thus the field of silylene chemistry is only about forty years old, a fact that is especially surprising when one realises the importance of these species to silicon chemistry. Silylenes are iso-electronic with carbenes, that is they have a six electron valency shell. However, the similarity with carbenes does not extend much beyond the electron configuration and a large part of the silylene chemistry that has received attention highlights this dissimilarity.

Perhaps the simplest approach to investigating the chemistry of silylenes is to pick out the differences in behaviour between them and their counterparts, carbenes. Carbenes are readily generated through the action of a strong base (e.g. LiR, or RMgBr, where R is an organic group) on halogenoalkyls. For example, production of dichlorocarbene proceeds easily using butyl lithium and trichloromethane.[57] However, the analogous reaction involving silicon species, to produce silylene, does not give the desired result. The actual products are those from the nucleophilic substitution of the halogen by R. Occasionally, the hydrogen itself may

be replaced. There are numerous methods to generate silylenes, one of the simplest being the dissociation of $SiCl_4$ to form $:SiCl_2$ caused by heating at 1073 K.[57] At this temperature, although the organic analogues do undergo thermal degradation to several products, carbenes are not involved.

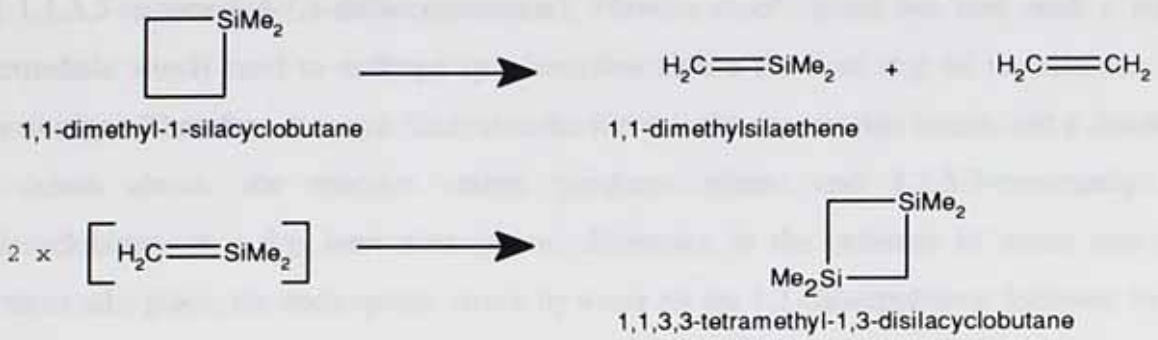
In addition to their formation, it is interesting to note that the behaviour of these two seemingly similar species is also quite different. Halogenocarbenes are known to dimerise, especially in the absence of electrophilic species with which they can react. Silylenes, however, do not dimerise under the same conditions but, again in the absence of electrophilic species, they do polymerise. Perhaps this particular fact is born of the same logic that gives silylenes longer lifetimes than the analogous carbenes. This is thought to be due to stabilisation in the case of silylenes from the empty *d* orbitals on the silicon. A similar stabilising effect could prolong the lifetime of intermediates in the polymerisation reaction, thus making it more likely that another silylene would be encountered and added to the chain before the active species decomposed back to an inert state. Indeed, it is reasonable to speculate that one of the factors that most separates silylene chemistry from carbene is the lifetime of the species involved. Perhaps another of the most important factors is likely to be the difference in polarisability and electronegativity between carbon and silicon. A good illustration of the difference in the relative stability of silylenes and carbenes is the isomerisation to alkene or cyclopropane type species (Scheme 11). Although this reaction is fast and efficient for carbenes, there is only approximately 5% conversion in the case of silylenes[57].



Scheme 11 Isomerisation of a carbene to olefin and cyclopropane

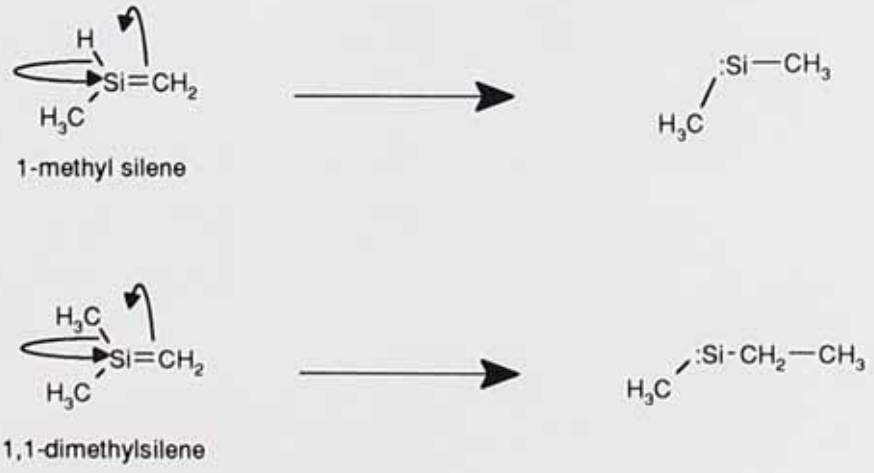
1,1-dimethyl-1-silacyclobutane

1,1-dimethyl-1-silacyclobutane provides one of the most effective procedures for generating 1,1-dimethylsilene (1,1-dimethylsilaethene) via its gas-phase pyrolysis (Scheme 12).



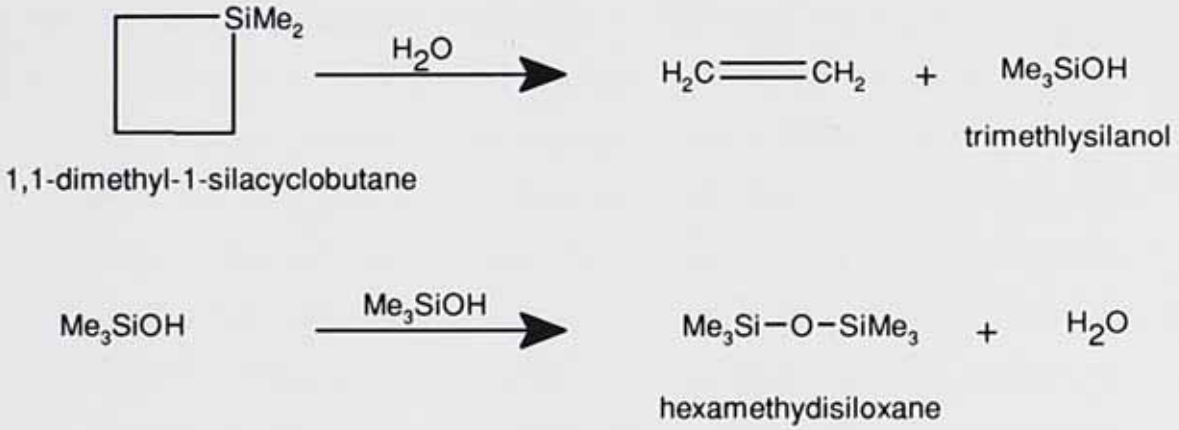
Scheme 12 Pyrolysis of 1,1-dimethyl-1-silacyclobutane in an inert atmosphere at 820 K

The silene produced in this way is stable enough to allow trapping and has been detected using mass spectrometric (MS) techniques.[58] Similar compounds which undergo an analogous pyrolysis, e.g. 1-methyl-1-silacyclobutane, give intermediates that are much harder to detect or trap due to their lower stability.[58] For example, 1-methylsilene is unstable with respect to the isomerisation to dimethylsilylene (Scheme 13) as the hydrogen shift is accomplished very easily. In the case of 1,1-dimethylsilene, however, the analogous isomerisation would require a methyl shift, a much slower process and hence a much longer lifetime for the species.[58]



Scheme 13 Isomerisation of 1-methylsilene and 1,1-dimethylsilene

The extended lifetime of the silene from 1,1-dimethyl-1-silacyclobutane allows a greater possibility of reaction in the pyrolysis mixture before decomposition. Although one can envisage a mechanism involving a diradical to give the products shown in Scheme 12 (ethene and 1,1,3,3-tetramethyl-1,3-disilacyclobutane), Flowers *et al*[59] point out that such a triplet intermediate would need to undergo spin inversion before the final step of the reaction, the dimerisation. Therefore, it seems likely that the silene is the species that reacts, not a diradical. As shown above, the reaction mainly produces ethene and 1,1,3,3-tetramethyl-1,3-disilacyclobutane in a dry, inert atmosphere. However, in the presence of water two new reactions take place, the nucleophilic attack by water on the 1,1-dimethylsilene followed by the condensation of the resulting trimethylsilanol to form hexamethyldisiloxane (Scheme 14).[59] Under these circumstances 1,1,3,3-tetramethyl-1,3-disilacyclobutane is not produced at all.



Scheme 14 Decomposition of 1,1-dimethyl-1-silacyclobutane in the presence of water

Experimental

1,1-dimethyl-1-silacyclobutane and hexamethyldisiloxane were handled under nitrogen and degassed, firstly by successive freeze, pump, thaw cycles and secondly by trap to trap distillation under vacuum. Products were identified by a combination of retention time comparison and by addition of an authentic sample of the material to the pyrolysis mixture after reaction, giving an enlarged peak on positive identification. Reactant pressures were measured into the reaction system, before injection, by a baratron linked to a Chell digital display.

Results and Discussion

Identification of pyrolysis products

Pyrolysis was carried out at temperatures from 730-860 K. At higher conversions (T > 810 K) four products were observed. However, at the lower temperatures fewer products were formed in sufficient quantities to be observed. The first product to be observed as the temperature was raised also had the smallest retention time. It seems likely that this product was ethene although this was not confirmed. Of the other three products detected, only one was identified. This product had the longest retention time of all the compounds detected and was found to be hexamethyldisiloxane (Scheme 14) by retention time comparison.

The fact that hexamethyldisiloxane was formed at all indicates that the system was subject to contamination with water. One product (the least retained) was almost certainly ethene, an inevitable 1,1-dimethyl-1-silacyclobutane pyrolysis product. Likely explanations for the other two products are that one is trimethylsilanol and the other 1,1,3,3-tetramethyl-1,3-disilacyclobutane. Previous workers[60] have observed that the complete suppression of 1,1,3,3-tetramethyl-1,3-disilacyclobutane production occurs only in the presence of a relatively large amount of water. Thus, partial suppression of the 1,1-dimethyl-1-silene dimerisation would result in some 1,1,3,3-tetramethyl-1,3-disilacyclobutane while trimethylsilanol must be formed in order for hexamethyldisiloxane to be produced.

The source of the water seems likely to have been a small leak from the atmosphere although it is possible that such a contaminant could have originated from another GC that was connected to a common gas-supply. Leak testing considered to be adequate had been performed at an earlier stage of the work and there was not enough time remaining to perform further tests once the above result had been obtained. However, it does highlight the problem for future work.

Order determination and rate variation with temperature

The order of reaction with respect to 1,1-dimethyl-1-silacyclobutane was measured at 810 K as 0.53 ± 0.04 (Figure 20). Such a figure is in agreement with the proposed mechanism (Scheme 14) in which each silacyclobutane decomposes to a trimethylsilanol which dimerises to the siloxane.

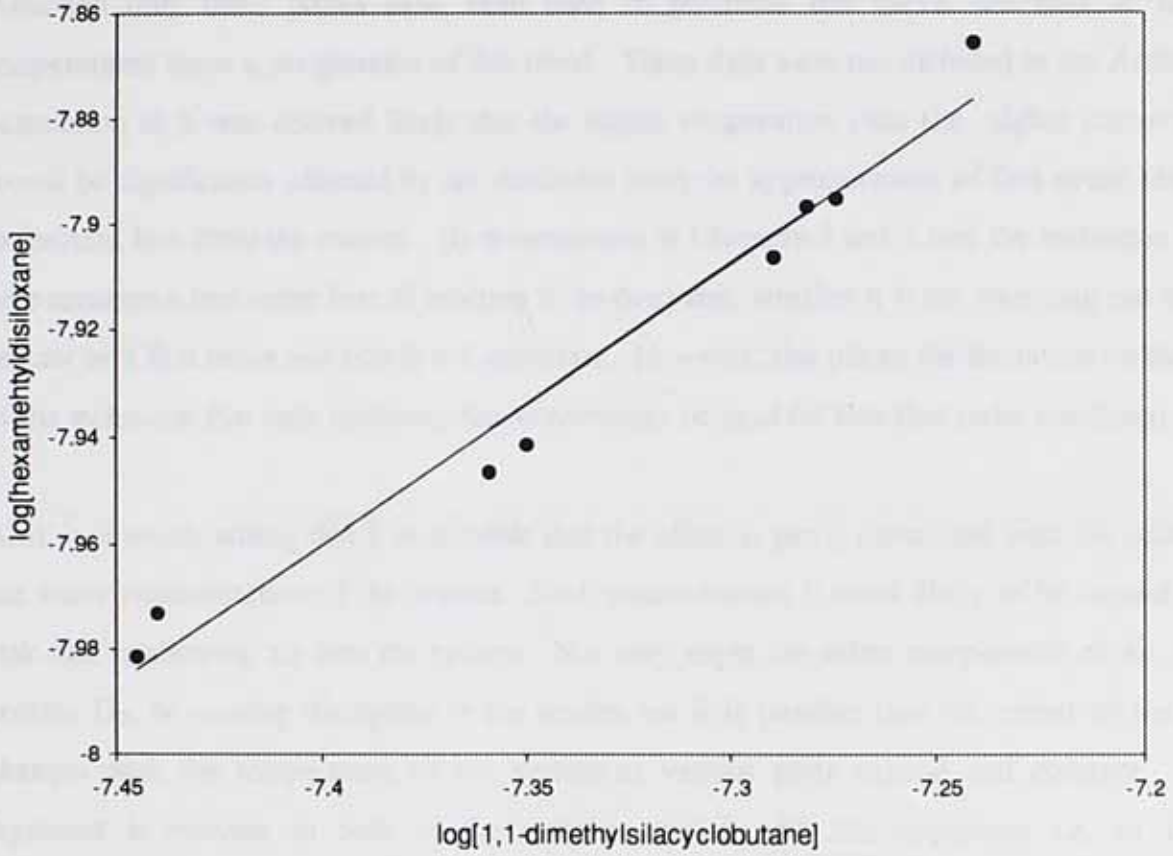


Figure 20 Order plot for the pyrolysis of 1,1-dimethyl-1-silacyclobutane at 810 K

Having established the reaction order, the temperature was varied over the range described above and the activation energy and $\log_{10}$(pre-exponential) factor measured as (351 ± 26) kJ mol^{-1} and 19.4 ± 0.1 respectively which seem unrealistically large. Although Arrhenius parameters are not given for the water contaminated reaction, the dry decomposition has a rate constant given by [59]:

$$k(\sec^{-1}) = 10^{15.8\pm0.2} \exp\left(\frac{-63.8 \pm 0.5}{RT}\right)$$ **Equation 27**

It can be seen that the pre-exponential factor is 3.5 orders of magnitude lower than that measured here, a discrepancy unlikely to originate solely from the difference in the reaction products. It is possible that the gradient calculated from the graph has been skewed, given the relatively few points and the non-linear nature of the data, giving rise to the strangely high Arrhenius parameters. The three points on the graph in Figure 21 do not lie on a straight line. Although only three points have been used to postulate this curve, the data at higher temperatures show a progression of this trend. These data were not included in the Arrhenius calculation as it was deemed likely that the higher temperature runs (i.e. higher conversion) would be significantly affected by the deviation from the approximation of first order kinetics in reactant loss from the reactor. (It is mentioned in Chapters 3 and 4 that the technique used here assumes a first order loss of reactant to be dominant, whether it is the sweeping out of the reactor or a first order reaction is not important. However, this places the limitation on the use of the technique that only relatively low conversions be used for non-first order reactions).

Also, it is worth noting that it is possible that the effect is partly connected with the cause of the water contamination of the system. Such contamination is most likely to be caused by a leak that is allowing air into the system. Not only might the other components of air, most notably O_2, be causing disruption in the results but it is possible that the extent of the leak changes with the temperature of the system as various parts expand and contract. This argument is relevant to both of the systems studied with this apparatus, i.e. to 1,1,1-trichloroethane pyrolysis and 1,1-dimethyl-1-silacyclobutane pyrolysis.

Finally, it would be advantageous to calculate the ratio of the amount of material entering the reactor to the amount leaving, thus ensuring that there are no losses that are unaccounted for. Although this precaution was taken for the TCE pyrolysis, in this case it was precluded within the time-scale of this project by the difficulty in obtaining pure samples of all the products for calibration of the GC.

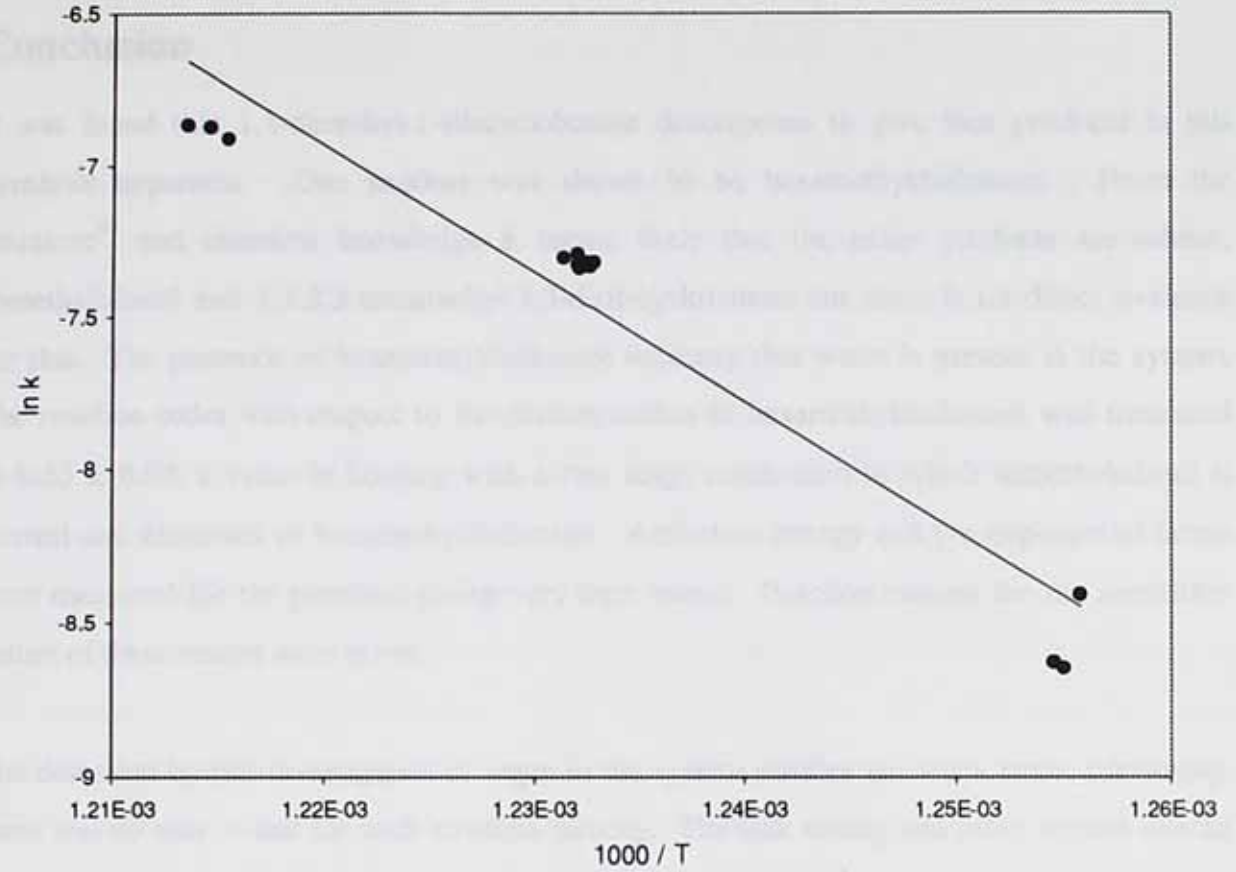


Figure 21 Arrhenius plot for the decomposition of 1,1-dimethyl-1-silacyclobutane

Conclusion

It was found that 1,1-dimethyl-1-silacyclobutane decomposes to give four products in this pyrolysis apparatus. One product was shown to be hexamethyldisiloxane. From the literature[59] and chemical knowledge it seems likely that the other products are ethene, trimethylsilanol and 1,1,3,3-tetramethyl-1,3-disilacyclobutane but there is no direct evidence for this. The presence of hexamethyldisiloxane indicates that water is present in the system. The reaction order with respect to the decomposition to hexamethyldisiloxane was measured as 0.53 ± 0.04, a value in keeping with a two stage mechanism in which trimethylsilanol is formed and dimerises to hexamethyldisiloxane. Activation energy and pre-exponential factor were measured for the pyrolysis giving very high values. Possible reasons for the unrealistic nature of these results were given.

The detection by this investigation of water in the system justifies the work since, previously, there was no way to test for such an effect directly. The leak testing and other routine checks that were made during the initial stages of this project are shown to be inadequate. The fact that they were so was beyond the available means of testing and, until unexpected results were obtained in the TCE pyrolysis, there was no reason to doubt them. It is unfortunate that such progress in identifying the cause of many of the problems with the experimental apparatus came so late in the project.

Further Work

Given additional time there are several things that should be done that would give valuable additional information about this system. However, as the investigation is of interest mainly in solving the difficulties encountered during the TCE pyrolysis experiments, the further work suggested here has been severely limited in scope.

Further investigation of the identity of the unknown products should be the first priority. If, as is supposed above, one of them is 1,1,3,3-tetramethyl-1,3-disilacyclobutane, kinetic investigation of the decomposition that yields it would give data for comparison with the literature values. Such a comparison would refine the picture of the extent and consistency of the amount of water in the system. Also, once identification and calibration had been completed a mass balance calculation should be performed for the system to be certain that no other processes, unaccounted for here, are taking place. More data collection on the contaminated system would be valuable in expanding the observed temperature range and giving an insight into the nature of the curve shape and its effect on the Arrhenius parameters.

In addition, much more exhaustive leak tests than those carried out during the construction stages of the project should be carried out on the system in an attempt to locate the source of the water contamination so that it can be rectified. Alternative explanations for the presence of water should be considered also, perhaps incomplete drying of the gas by the filtration system, and tests performed to check these hypotheses.

Chapter 6 : Conclusions and Further Work

Investigation of the pyrolyses of organochlorine compounds of environmental interest

The pyrolysis of 1,1,1-trichloroethane was chosen as a representative example of reactions of this type and as an analogous process to the thermal decomposition of 1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane (DDT). The data from the investigation were inconclusive and not totally reproducible. The identification of the major organic product of the pyrolysis as 1,1-dichloroethene (CH_2CCl_2) agreed with the literature report.[45,48] However, the measurement of the order of the reaction was not as straightforward. It was found that the order varied with the temperature of the reaction. Calculation of parameters such as activation energy and the Arrhenius pre-exponential factor also gave a range of values with varying errors. Much of the uncertainty observed in these data can be ascribed to the apparent temperature dependence of the reaction order. Rate constant calculation is very sensitive to the order of reaction for this technique and small changes in the order due, in part, to temperature fluctuations could be to blame for much of the irreproducibility of these results.

However, although the variation in the measured rate constant is explained partly by the change in reaction order, this is not sufficient to account for the magnitude of uncertainty. Also, it is necessary to explain the variation in the reaction order itself. One explanation that goes some way to rationalising the results satisfactorily is the presence of two competing processes within the reactor. It is known that organochlorine processes in general and this one in particular are very susceptible to heterogeneous effects and that prolonged conditioning of the experimental vessel is necessary before investigation of the reaction can be assumed to be only homogeneous.[39,61] Such heterogeneous processes often have reaction orders with respect to the reactant approaching zero as they are diffusion controlled. The combination of the heterogeneous process and the homogeneous process reported in the literature[45,46] would give a range of reaction orders and a range of activation energies in the overall process, depending on which mechanism was dominating at any particular moment. The results are consistent with

this behaviour. Although attempts were made to 'condition' the reactor (through the injection of 'liquid' pulses through a septum modification directly into the reactor ~20 mm^3 TCE), as is usual in the field, the amounts of reactant used in the reaction was so small that the effectiveness of such a procedure was doubtful. Also, even a 'conditioned' reactor might be expected to have a small component of heterogeneity. Such a minor process could account for a significant portion of the results seen for the micro-injections used in this work.

In addition to the heterogeneous / homogeneous competition, it was found that significant levels of water were present in the reaction system, probably from the atmosphere, by the use of a test reaction, the pyrolysis of 1,1-dimethyl-1-silacyclobutane. The presence of water and the implied presence of atmospheric oxygen would have had unknown effects on the 1,1,1-trichloroethane reaction. One can speculate that the levels of atmospheric contamination would fluctuate thus increasing the variation in results. Also, it has been reported that exposure of a conditioned vessel wall to oxygen for as little as 15 minutes has required the complete re-conditioning of the reaction vessel.[61] Continuous oxygen contamination, even at very low levels, would be expected to make the conditioning of the vessel impossible, regardless of any other problems.

Thus, the investigation of the pyrolysis of 1,1,1-trichloroethane was hampered by many other considerations. Some useful data were collected, especially where the results corroborate other work using this new technique.

Development of the technique of pulse stirred-flow investigation to organochlorine pyrolyses

This work has been very useful in identifying and overcoming some of the problems inherent in using the pulse stirred-flow technique for processes of this type. Three areas of interest can be identified. The detector system must be very reliable and sensitivity occasionally has to be sacrificed to obtain this. The unreliability of the electron capture detector (ECD) is connected, in part, with the contamination of the gas system, presumably through an atmospheric leak.

ECDs are sensitive to oxygen and water levels in the gas flows. The suspected oxygen contamination was a problem not only because of the processes that it might initiate itself but also because of its effect on the vessel walls. The reactivity of the vessel walls and the possibility of heterogeneous processes must be considered the most serious problems. The possible presence of oxygen and water at low levels might be negligible if the vessel walls could be permanently desensitised to reaction.

Alternative methods of desensitising the vessel walls were investigated, e.g. the use of silanising agents or the coating of the internal surface of the vessel with PTFE, but they all suffer from the same problem, that of a lack of thermal stability. None of the materials or reagents investigated were stable above 600 K and most were useless above 570 K. The solution might be to use vessels formed of a different material entirely, perhaps an inert ceramic.

Finally, to speculate a little, the dependence of the rate constant on the reaction order that is pointed out above is decidedly non-linear. Non-linear processes are known to be prone to unpredictable behaviour. Perhaps it is not surprising that the results seemed strange given such unpredictability especially when coupled with the flow control method for this system. The original pulse stirred-flow technique[43,42] utilised mass flow controllers to regulate the amount of gas through the system. Such devices are not prone to the sort of feedback behaviour that a system wholly dependent on pressure regulation can experience. Both the rate constant / reaction order dependence and the possibility of feedback behaviour from the pressure regulation are systems likely to be very susceptible to chaotic behaviour. To eliminate this possibility future work should utilise a more reliable flow control and, as stated above, the problem of heterogeneous reaction must be overcome.

In conclusion, the pulse stirred-flow technique is potentially useful for work in this field, if the problems discussed above can be overcome. Solutions to those problems should be sought as the advantages that originally led to the choice of this technique are still valid, i.e. low cost, simplicity and speed. Not only has this work identified the problems likely to be encountered by future workers, but some useful tools in testing the effectiveness of possible solutions have been developed, e.g. the use of the 1,1-dimethyl-1-silacyclobutane pyrolysis to establish the integrity of the system from contaminants such as water.

Further Work

Although much of the kinetic work introduced in the Aims section remains to be done and could be completed at a later date, the first priority for further work must be the development of the investigative apparatus. The main requirements are for an inert reaction vessel, a completely leak tight gas supply system (preferably with mass flow control) and a more reliable detector system. Perhaps the flame ionisation detector (FID) could be coupled with a thermal conductivity detector (TCD) which is sensitive to a very wide spectrum of chemicals and does not suffer greatly from reliability problems.

Until the solution of the problems stated above, it would be fruitless to pursue any kinetic investigations further. However, once the reliability of the apparatus had been established (using the 1,1-dimethylsilacyclobutane reaction), it would be wise to repeat the work on 1,1,1-trichloroethane and, as intended, use this as a basis for the investigation of more complicated materials and processes, such as the pyrolysis of 1,1,1-trichloro-2,2-bis(4-chlorophenyl)ethane (DDT). Furthermore, it would be interesting to attempt to extend the applicability of the technique to other types of organochlorine compounds, also of environmental significance, e.g. chlorofluorocarbons (CFC's).

Appendix A : Detectors

Flame Ionisation Detector (FID)

The flame ionisation detector is one of the most sensitive, reliable detectors available and is popular because of the wide spectrum of organic materials to which it is sensitive and for its large linear range (10^7).[62] The principle upon which it relies is that a flame produced by burning hydrogen in air has a low electrical conductivity which increases by several orders of magnitude when organics are present in the fuel gases. Two explanations are offered for this phenomenon. Current flows either via charged carbon particles formed by the combustion / pyrolysis process in the flame or by ionised organic molecules, also produced by the exothermic combustion. Thus the detector mixes fuel gases and column effluent and measures the conductivity of the flame.

The greatest sensitivity has been found if the flame jet is made the cathode with the anode surrounding it. Temperature is also important to the detector's response and most configurations have the detector body inside a thermostated oven. It is best if a separate dedicated heated zone is used, rather than the practice of placing the detector in the column oven, as temperature programming would become difficult in such circumstances.

There are many successful FID designs but a good general purpose configuration is shown in Figure 22. The detector body contains a metal block in which to mix the hydrogen and column effluent and an electrically shielded cavity filled with air in which the hydrogen mixture burns. Also present are the two electrodes across which a constant potential difference is maintained while the current is measured. The output of the detector is proportional to the number of ions produced in the flame which, for a given number of moles of an organic material, is approximately proportional to the percentage carbon per molecule. The sensitivity for a given percentage of carbon varies slightly between homologous series but is reliable within a series. It is wise, however, to calibrate the detector for every compound quantified to obtain the best results.

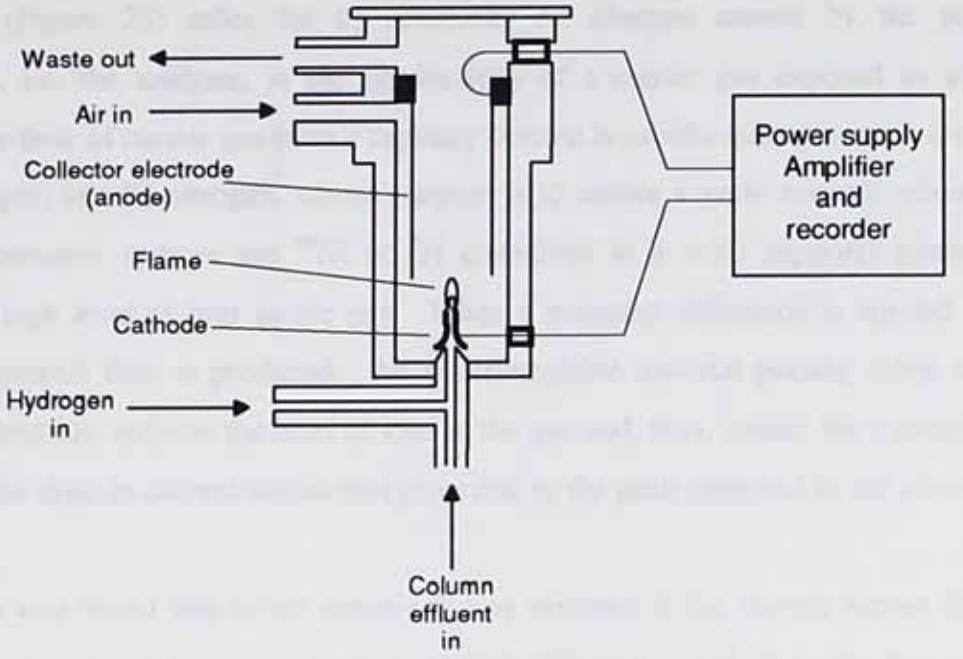


Figure 22 Schematic representation of a flame ionisation detector (FID)

Electron Capture Detector (ECD)

The ECD (Figure 23) relies for its sensitivity on changes caused by the presence of 'impurities', i.e. the analytes, in the conductivity of a carrier gas exposed to a β-particle source. The flow of carrier gas from a capillary column is usually quite low so it is mixed with a make-up gas, usually nitrogen, whose purpose is to ensure a great enough volume that the radiation (common sources are ^{60}Ni or ^{3}H embedded in a solid support) gives rise to a sufficiently high level of ions in the gas. When a potential difference is applied across the detector a current flow is produced. An electronegative material passing along the column enters the detector, reduces the level of ions in the gas and, thus, causes the current output to drop. It is the drop in current output that gives rise to the peak observed in the chromatogram.

However, it was found that better sensitivity was obtained if the current across the detector was kept constant and the variation in potential difference required to do this used as the output. Modern detectors have improved on this by using a pulsed potential difference of a constant amplitude. The frequency of the pulse is varied so that the current is constant and the frequency becomes the output. One of the reasons for the improvement in sensitivity and accuracy thus gained is the greater ease with which a frequency, as compared with a potential difference, may be measured electronically.

One of the ECD's main advantages is its enormous sensitivity (as little as 10^{-19} mol of some compounds have been detected[44]) and selectivity to electrophilic compounds. However, this is also a disadvantage as it makes the detector very sensitive to contamination by minute amounts of chemicals such as oxygen, water and volatile halogenated compounds, e.g. leaching of monomeric residues trapped in polymeric matrices, polytetrafluoroethene used in tubing for example. To make use of this detector, great care must be taken to remove any traces of contaminants from the carrier and make-up gases.

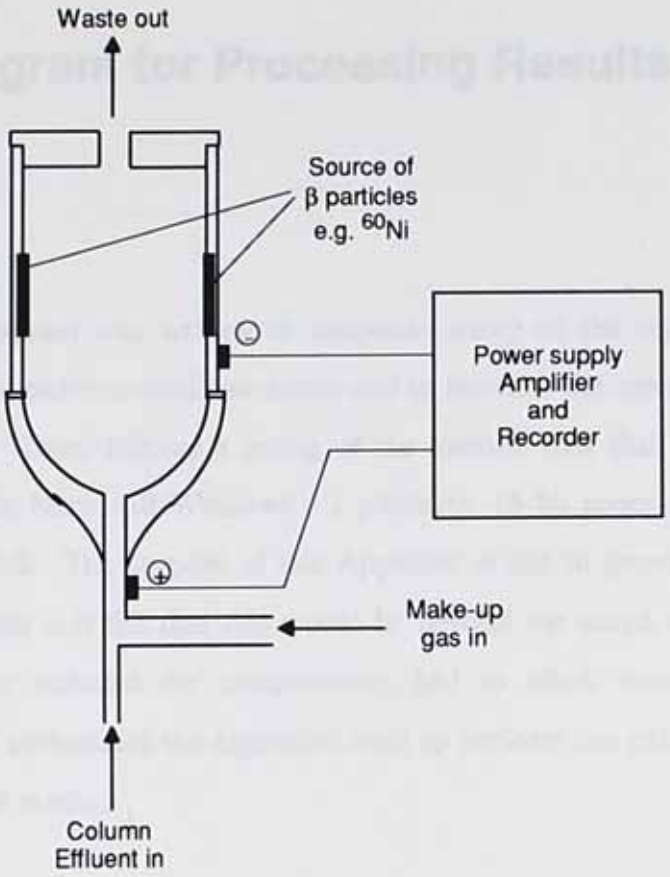


Figure 23 Schematic Representation of an Electron Capture Detector (ECD)

Appendix B : Program for Processing Results

Introduction

As part of this project, a program was written to automate many of the repetitive tasks involved in processing results in order to minimise errors and to increase the speed with which analyses could be performed. There follows a listing of the various files that make up the program. The program uses the Microsoft Windows 3.1 platform, 16 bit processing and was written using Borland C++ v.4.2. The purpose of this Appendix is not to provide a detailed explanation of the program since it is felt that this would be beyond the scope of this thesis. The program listing has been included for completeness and to allow someone with a knowledge of Borland C++ to understand the algorithm used to perform the calculations that were a part of the processing of results.

The name of each file heads its listing. Those files having the ending ".h" are header files containing datatype declarations and other preliminary pre-compiler directions. Those files ending ".cpp" are files containing code, each one comprising a module of function and / or class definitions (as opposed to declarations). The file entitled "Kinet1.cpp" holds the 'OwlMain' function and the actual program itself. The other code files are modules subsidiary to this file.

The data from each experiment were entered through 3 dialogue boxes, including that used to enter data about the reactor size and the values to be used for various constants. Processing took place according to various commands entered through key combinations or via the menu. The data could be saved for further analysis later, by this program, or output in a format suitable for input into a spreadsheet for graphical work. Generally the results were output into a spreadsheet and graphs drawn using that package.

File List

Data_def.h

Kinclass.h

Kinet.h

Kinetcls.h

Kinetb.h

Kinetrc.h

Macextr.h

Kinet1.rc

Consts.cpp

Extr.cpp

Genfunct.cpp

Kinet1a.cpp

Kinetcls.cpp

Kinet1.cpp

Kinet1.def

Data_def.h

```
// file containing defs for data constants
#if !defined _data_ref_h__
#define _data_ref_h__
// sample loop volume in m3
#define SAMPLELOOP 0.25e-6
#define R   8.314          // gas constant
// temp conversion factor deg C -> K (add it)
#define CinK 273
// torr -> Pa conversion
#define TORRtoPA (101325/760)  // needs updating to better
accuracy
#define ATMOSPRESSpsi 14.696   // atmospheric press in psi
#endif
```

Kinclass.h

```
// this header has incomplete declarations of all classes created for this
program
// it also has template definitions needed
// classes
class TConfigData;
class TCalTransBuffer;
class TInputBoxTransferBuffer;
class TConfigTransferBuffer;
class TProdTransferBuffer;
class TReactTransferBuffer;
class TProductReactantCal;
class TWinApp;
class TFileType;
class TInputBox;
class TProdDialog;
class TConfigDialog;
class TReactDialog;
class TMainWindow;
class TKinDataStruct;
class KinetData;
class TCalDataStruct;
class TCalData;
```

Kinet1.h

```
#if !defined __KINET1_H
#define __KINET1_H     // prevent multiple includes
#define RES 255        // for TFileType class
#define BINARY 4
#define DELIMATED 8
#define NONE  0
#define ALL  2
#define REACTORPRESS
                       1L          // defines for ConfigData data
checks
#define FLOW                   2L
#define MEASTEMP          4L
#define REACTORVOL         8L
#define GASCONST          16L
#define DETECTOR          32L


#include <mem.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>
#include <iostream.h>
#include <math.h>
#include <fstream.h>
#include <cstring.h>
#include "kinetrc.h"
#include "kinclass.h"

const int MaxEditLen = 80;
const int BufLen    = 2048; // length of buffer used to build output
string


// structure for configuration data
class TConfigData    // to hold the config data from the dialog box
{
public:
double ReactorPress; // psi
 double FlowRate;   // cm3 s-1  measured at atmos temp and press
 double MeasTemp;   // temp at which flow measured
double ReactorVol; // cm3
double GasConstant; // value of the gas constant, R
char Detector[30];

TConfigData();
long CheckData(); // checks for the presence of the data
inline void Set(TConfigTransferBuffer& c);
ofstream& Save(ofstream& os, char Delimator, ios::open_mode
mode);
// for saving to file;  most functions have 'manually' saved this to binary
but
// a function is needed for text mode
TConfigData& operator=(double num);
};



// Transfer buffers ------------------------------
class TCalTransBuffer
 {
public:
char Name[MaxEditLen];
char PressInj[MaxEditLen];
char TempInj[MaxEditLen];
char PeakArea[MaxEditLen];

 void Set(TCalDataStruct& b);
 TCalTransBuffer()
 { memset( (char*) this, 0, sizeof(TCalTransBuffer) ); }
 TCalTransBuffer(TCalDataStruct& b)
            {
Set(b);
            }
};


class TInputBoxTransferBuffer
{
public:
char InputData[MaxEditLen];
TInputBoxTransferBuffer()
{ memset(this, 0, sizeof(TInputBoxTransferBuffer) ); }
};


class TConfigTransferBuffer
{
public:
char FlowRate[MaxEditLen];   // cm3 s-1  measured at atmos temp
and press
char ReactorPress[MaxEditLen]; // psi
char MeasTemp[MaxEditLen];   // temp at which flow measured
char ConstR[MaxEditLen];
char Detect[MaxEditLen];
char ReactVol[MaxEditLen];

TConfigTransferBuffer()
{ memset(this, 0, sizeof(TConfigTransferBuffer) ); }
inline void Set(TConfigData& c);
};

class TProdTransferBuffer
{
public:
char CompName[MaxEditLen];
char ChemForm[MaxEditLen];
char RetentTime[MaxEditLen];
char PeakArea[MaxEditLen];
char CoeffC[MaxEditLen];
char CoeffM[MaxEditLen];
TProdTransferBuffer()
{ memset(this, 0, sizeof(TProdTransferBuffer) ); }
void Set(TKinDataStruct& k);
};

class TReactTransferBuffer
{
public:
char CompName[MaxEditLen];
char ChemForm[MaxEditLen];
char DatePerformed[MaxEditLen];
char RetentTime[MaxEditLen];
char PeakArea[MaxEditLen];
char TrapTime[MaxEditLen];
char ReactorTemp[MaxEditLen];
char CoeffC[MaxEditLen];
char CoeffM[MaxEditLen];
char InjPress[MaxEditLen];
char InjTemp[MaxEditLen];
TReactTransferBuffer()
{ memset(this, 0, sizeof(TReactTransferBuffer) ); }
void Set(TKinDataStruct& k);
};
#endif // __KINET1_H
```

Kineticls.h

```
#if !defined __kineticls_h__
#define __kineticls_h__

#include <windows.h>   // definition for BOOL
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <fstream.h>
#include <strstrea.h>
#include "data_def.h"
#include "kinclass.h"
#include "kinet1.h"
#ifdef MaxEditLen
#define MAX (MaxEditLen+1)
#else
#define MAX 82
#endif

#define INJPRESS                    1  // constants for
missing essential data
#define INJTEMP                     2
#define REACTORTEMP         4
#define PRODCOEFFM          8
#define REACTCOEFFM         16
#define PRODPEAKAREA        32
#define REACTPEAKAREA       64
#define  DATE                                   1     //
constants for missing non-essential data
#define TRAPTIME                    2
#define PRODRETENT                  4
#define REACTRETENT         8
#define PRODNAME                    16
#define PRODFORM                    32
#define REACTNAME                   64
#define REACTFORM                   128

enum FIELDS { REACT, PROD, TOTAL };
enum PROB { UNKNOWN, STRING_TOO_LONG,
DISK_ERROR, UNABLE_TO_OPEN };

// error class for exceptions
class FileReadErr
 {
public:
char* str; // to hold name of problem field
long FilePointer;  // position in file reached
PROB fault;   // nature of problem
KinetData* tmp; // to pass object for deleting
TCalData* obj;  // to pass object for deleting

FileReadErr()
          {
str = NULL;
FilePointer = 0;
fault =  UNKNOWN;
tmp = NULL;
          }

FileReadErr(const char* s, long p, PROB f, KinetData* t = 0)
          {
Set( s, p, f, t);
          }

FileReadErr(const char* s, long p, PROB f, TCalData* t)
          {
Set( s, p, f, t);
          }

~FileReadErr(); // body in kineticls.cpp

void Set(const char* s, long p, PROB f, KinetData* t = 0)
          {
str = new char[strlen(s)+1];
strcpy(str, s);
FilePointer = p;
fault = f;
tmp = t;
          }

void Set(const char* s, long p, PROB f, TCalData* t = 0)
          {
str = new char[strlen(s)+1];
strcpy(str, s);
FilePointer = p;
fault = f;
obj = t;
          }
 };
/*_____________________________________
Calibration Data structure
_____________________________________*/
struct TCalDataStruct
 {
char Name[MaxEditLen];
double pressinj; // torr
double amount;  // mol
double tempinjC; // deg C
double tempinjK; // K
double peakarea; // uV s
TCalDataStruct()
{ Reset(); }
void Reset()
{ Name[0] = pressinj = amount =tempinjC = tempinjK = peakarea = 0; }
void Set(TCalTransBuffer& b);
void Sprintf(ostrstream& stream, FIELDS f);
ofstream& Save(ofstream& os, char Delimator, ios::open_mode
mode);
ifstream& Load(ifstream& is, char Delimator, ios::open_mode mode);
 };

class TCalData
 {
protected:
TCalDataStruct Data;
public:
TCalData()
{ Data.Reset(); }
TCalData(TCalTransBuffer& b);
TCalDataStruct& data()
{ return Data; }
void Set(TCalTransBuffer& b);
BOOL operator ==(const TCalData& other) const
          {  return BOOL( &other == this ); }
          // prints set fields into str
void Sprintf(char* str, int sz = MAX, FIELDS f = TOTAL );
ofstream& Save(ofstream& os, char Delimator, ios::open_mode
mode);
ifstream& Load(ifstream& is, char Delimator, ios::open_mode mode);
 };


/*_____________________________________
k = (B) / (A0) tau      (where tau is the residence time for the reactor)
Independent of the degree of conversion
This only depends on there being "perfect mixing"
*/
struct TKinDataStruct
 {
// one-off experimental data
char DatePerformed[MAX];
  double InjPress;   // torr
  double InjTempC;    // C
  double InjTempK;    // K
  double AmountInj;  // mol
```

```
 float TrapTime;  // min
 double MassBalance;   // ratio of amount in / amount out
// product data
double ProdRetentTime; // min
 double ProdPeakArea;  // uV s
 double ProdCoeffM;   // approx 1E+11
double ProdCoeffC; // usually 0
char ProdName[MAX];
char ProdChemForm[MAX];
// Reactant Data
 double ReactRetentTime;  // min
 double ReactPeakArea;   // uV s
 double ReactCoeffM;    // approx 1E+11
double ReactCoeffC; // usually 0
char ReactName[MAX];
char ReactChemForm[MAX];
// Non - constant Reactor Data
 double AdjustedFlowRate;   // cm3 s-1
 double ReactorTempC;       // C
 double ReactorTempK;       // K
// Results calculated
double RateConstant;  // units depending on order
double ReactAmount;  // mol calc from Peak areas and sensitivities
 double ProdAmount;  // mol calc from Peak areas and sensitivities
 double Tau;       // residence time of reactor / s
 double Order;      // order of the reaction
 double logReact;    // log of the amount of reactant detected for order
plots
 double logProd;     // log of the amount of product detected for order
plots
 double lnRateConst;  // natural log of rate constant
 double InverseTemp;       // 1/T  for plot against lnRateConst
TKinDataStruct()
{ Reset(); }
void Set(TProdTransferBuffer& Prod);
void Set(TReactTransferBuffer&  React);
ofstream& Save(ofstream& os, char Delimator, ios::open_mode
mode);
void Reset();
};

class KinetData  // should be one object for each experiment (injection)
{
public:
static TConfigData Config; // to allow easy sharing of data with all
kinetic data objects
enum save_flag { binary, delimated};
protected:
// flag (s)
BOOL Modified;  // whether or not anything has been changed since
the
// last saving
save_flag mode;  // how to save or open a file
TKinDataStruct Data;
public:
KinetData(save_flag m = binary);
KinetData(TProdTransferBuffer&  Prod, save_flag m = binary);
KinetData(TReactTransferBuffer&  React, save_flag m = binary);
 KinetData(TReactTransferBuffer&  React,   // reactant info

           TProdTransferBuffer&  Prod,  // product info
                                          save_flag m =
binary);
BOOL operator ==(const KinetData& other) const
          {     return BOOL( &other == this ); }   // for built in data
containers etc
 void modified(BOOL m) {Modified = m;}
 void Mode(save_flag m) {mode = m; } // set open / save flag
// prints set fields into str
void Sprintf(char* str, int sz = MAX, FIELDS f = TOTAL ); // prints set
fields into str
void Reset();
// these two call CalcAmounts() so that changes cause their own
updates
void Set(TProdTransferBuffer& prod);
void Set(TReactTransferBuffer&  react);
// does not call CalcAmounts() as an order change does not affect
these calculations
void Set(double order);
TKinDataStruct& data()  // allows access to protected data
{ return (TKinDataStruct&) Data; }
long CheckEssData(); // check for presence of data essential to
calculation
// for some pieces of data eg CoeffC  zero is accepted
long CheckOtherData(); // checks for presence of data not essential to
calculation
// but that might be useful for the record
int CalcAmounts(); // calc's those quantities that can be dome
immediately
ofstream& Save(ofstream& os, char Delimator);
ifstream& Load(ifstream& is, char Delimator);
//friend ostream& operator << (ostream& os, KinetData& data);
//friend istream& operator >> (istream& is, KinetData& data);
 };
#endif  // __kinetcls_h__
```

Kinet1b.h

```
#if !defined __kinet1b_h
#define __kinet1b_h
extern char Delimator; // variable to hold character to delimate export
files with
extern const char Version[];
extern const char ConfigTitleText[];
extern const int BufSize; // size of buffer used in import function to
check format
extern const double DefReactorVol; // default reactor volume used at
startup
extern const double DefGasConstant; // default gas constant
extern const char DefDetector[]; // default detector type
// opensave dialog text for filters etc
extern const char ExpExt[]; // default extension for import / export files
extern const char OpenSaveFilter[];
extern const char ImportExportFilter[];
extern const char OpenSaveExt[]; // default extension for data normal
files
// text for title in Kinetic data view. See KinetData::Sprintf in
// Kineticls.cpp for order of output
extern const char KineticTitleText[];                                    // 1
tab
#define KineticNumTabs 30
// tab positions in characters. Used later to calculate coordinates for
actual tabs
extern const int KinetictabChar[30];
// text for Product/Reactant view title
extern const char ProdReactTitleText[];
#define ProdReactNumTabs 5
// tab positions in characters. Used later to calculate coordinates for
actual tabs
extern const int ProdReacttabChar[5];
extern const char LinRegressTitle[];
extern const int LinRegrTitTab[5];
#define NumRegrTitTab 5
// title of arrhenius fields
extern const char TArrheniusTitle[];
extern const int TArrheniusTitTab[3];
#define NumTArrheniusTitTab 3
#endif
```

Kinetrc.h

```
#if !defined _KINETRC_H
#define KINET1_ICON 5000
#define _KINETRC_H
#define ID_CAL_TEMP 602
#define ID_CAL_PRESS 601
#define ID_CAL_PEAK 600
#define ID_CAL_NAME 603
#define DLG_INPUT 600
#define CM_EXPORT 511
#define CM_IMPORT 510
#define CM_DEBUG_INFO 550
#define CM_EDIT_ORDER 101
#define CM_CALC_ACTENERGY 802
#define CM_DATA_NEWPROD 1011
#define CM_DATA_NEWREACT 1010
#define CM_CALC_CALPROD 811
#define CM_CALC_CALREACT 810
#define CM_CALC_ORDER 801
#define CM_CALC_AMOUNTS 800
#define CM_VIEWKINETIC_DATA 701
#define CM_VIEW_REACTANT 703
#define CM_VIEW_PRODUCT 702
#define CM_FILE_SAVE 502
#define CM_FILE_CLOSE 503 // menu commands
#define CM_FILE_OPEN 507
#define CM_FILE_SAVEAS 500
#define CM_CONFIG 501
#define CM_DATA_NEWKIN 1000
#define CM_DATA_DELETE 1002
#define CM_DATA_EDIT 1001
#define IDC_INPUT_EDIT 601
#define ID_PROD_DLG 304
#define ID_REACT_DLG 804
#define ID_CAL_DLG 310
#define ID_CONFIG_DLG 301
#define ID_KINET_MENU 302
#define ID_CONF_FLOW 406
#define ID_CONF_PRESS 405
#define ID_CONF_MEASTEMP 404
#define ID_CONF_DETECT 402
#define ID_CONF_R 401
#define ID_CONF_VOL 403
#define ID_PROD_RETENT 209
#define ID_PROD_COEFFC 205
#define ID_PROD_COEFFM 206
#define ID_PROD_PEAKAREA 207
#define ID_PROD_FORM 208
#define ID_PROD_NAME 213
#define ID_REACT_COEFFM 802
#define ID_REACT_COEFFC 801
#define ID_REACT_PEAKAREA 807
#define ID_REACT_REACTTEMP 808
#define ID_REACT_TEMPINJ 810
#define ID_REACT_RETENT 809
#define ID_REACT_TRAP 811
#define ID_REACT_INJPRESS 805
#define ID_REACT_DATE 803
#define ID_REACT_CHEMFORM 800
#define ID_REACT_NAME 806
#endif
```

Macextr.h

```
// header for function writing macros
// function for extracting data from container objects
// must be in a position to know what TLongArray is when this macro is
used
#define KINETICDATA 1
#define CALIBRATIONDATA 2
// macro declares correct function declaration given the data type and
the function name
```

```
#define DECLARE_EXTRACT(function_name, data_type)  \
long Extract_## function_name(double* X, double* Y, \
data_type& data, TLongArray& selected);
// data_type is the name of the container object type containing the data
// X_name and Y_name are the name of the fields to extract into X and
Y
#define Extract_write(function_name, data_type, X_name, Y_name)  \
long Extract_## function_name(double* X, double* Y, \
data_type& data, TLongArray& selected) \
{
if( !X || !Y) return 0;
long lines = selected.GetItemsInContainer();
TLongArrayIterator i(selected);
double* x = X;
double* y = Y;

while(i)
 {
 *x = (data[i.Current()]->data()).X_name;
 *y = (data[i++]->data()).Y_name;
 x++;
 y++;
 }
return lines;
}

Genfunct.h
#if !defined __GENFUNCT_H
#define   __GENFUNCT_H
/*
#if !defined __WINDEF_H
#include <windef.h>      // for BOOL

#endif
*/
/*-------------------------------------------------------
Data Types Needed
-------------------------------------------------------*/
// structure for passing coefficients and error data
struct LinRegress
 {
// y = mx + C
double m;  // coeff m of x
double c;  // coeff c
double CorrCoeff;  // correlation coefficient
double CorrCoeff_sqr; // squared to avoid negative values
double rms; // root mean squared deviation
double PerCentErrorM;  // %-age error in m

long n;  // number of points
LinRegress();
void Sprintf(char* str, int Len);
void reset();
 };

/*-------------------------------------------------------
string functions
-------------------------------------------------------*/
// replaces the string find with replace in str.  str taken to be to first \x0
// returns number of replacements that have taken place
int CharCount(char* s, int ch);
int CharReplace(char* s, const char* find, const char* r);
int CharReplace(char* s, const char* find, const char r);
// can be used to replace one character
BOOL NumTruncate(char* input); //truncates at first non-digit
(excluding '.')
// returns TRUE if there is any string remaining
/*-------------------------------------------------------
stats functions
-------------------------------------------------------*/
double square(double d);
LinRegress LinearRegress(double* X, double* Y, long NumPoints);
#endif
```

Kinet1.rc

```
#include <owl\window.rh>
#include <windows.h>
#include <owl\inputdia.rc> // input dialog box
#include "kinetrc.h"

ID_REACT_DLG DIALOG 9, 16, 293, 169
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU
CLASS "BorDlg_Gray"
CAPTION "Reactant"
FONT 10, "Arial"
{
EDITTEXT ID_REACT_INJPRESS, 71, 77, 36, 11
EDITTEXT ID_REACT_TEMPINJ, 192, 77, 36, 11
EDITTEXT ID_REACT_TRAP, 71, 97, 36, 11
EDITTEXT ID_REACT_REACTTEMP, 192, 97, 36, 11
EDITTEXT ID_REACT_RETENT, 71, 118, 36, 11
EDITTEXT ID_REACT_PEAKAREA, 192, 118, 51, 11,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_REACT_NAME, 75, 4, 68, 11, ES_AUTOHSCROLL
| ES_NOHIDESEL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_REACT_CHEMFORM, 75, 21, 68, 11,
ES_AUTOHSCROLL | ES_NOHIDESEL | WS_BORDER |
WS_TABSTOP
EDITTEXT ID_REACT_DATE, 214, 4, 68, 11,
ES_AUTOHSCROLL | ES_NOHIDESEL | WS_BORDER |
WS_TABSTOP
EDITTEXT ID_REACT_COEFFM, 99, 51, 53, 13
EDITTEXT ID_REACT_COEFFC, 208, 51, 53, 13
CONTROL "Button", IDOK, "BorBtn", BS_DEFPUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 107, 140, 33, 21
CONTROL "", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 153, 140, 33, 21
RTEXT "Injection Temp", -1, 136, 78, 53, 8
RTEXT "Compound Name", -1, 11, 5, 59, 9
RTEXT "Reactor Temperature", -1, 146, 95, 43, 16
RTEXT "Trap Time", -1, 28, 98, 39, 9
LTEXT "min", -1, 114, 99, 12, 8
LTEXT "torr", -1, 114, 78, 14, 9
LTEXT "deg C", -1, 248, 78, 20, 9
CTEXT "y = m x + c", -1, 14, 53, 42, 10, SS_CENTER |
WS_BORDER | WS_GROUP
RTEXT "Coeff m :", -1, 64, 53, 31, 9
RTEXT "Coeff c :", -1, 173, 53, 31, 9
RTEXT "Peak Area", -1, 150, 119, 39, 8
LTEXT "deg C", -1, 248, 98, 21, 10
RTEXT "Retention Time", -1, 7, 118, 60, 10
LTEXT "min", -1, 114, 119, 14, 8
CONTROL "Detector Sensitivity", -1, "BorShade", BSS_GROUP |
BSS_CAPTION | BSS_LEFT | WS_CHILD | WS_VISIBLE, 9, 38,
261, 31
RTEXT "Chemical Formula", -1, 36, 18, 34, 18
RTEXT "Press Injected", -1, 16, 78, 51, 8
RTEXT "Date Performed :", -1, 156, 5, 55, 8
LTEXT "uV s", -1, 248, 119, 14, 8
CONTROL "", -1, "BorShade", BSS_HDIP | BSS_LEFT |
WS_CHILD | WS_VISIBLE, 2, 133, 289, 5
}
ID_CONFIG_DLG DIALOG 39, 66, 208, 167
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CLASS "BorDlg_Gray"
CAPTION " Parameter Configuration"
FONT 8, "MS Sans Serif"
{
EDITTEXT ID_CONF_VOL, 75, 11, 65, 13, ES_AUTOHSCROLL |
WS_BORDER | WS_TABSTOP
EDITTEXT ID_CONF_R, 75, 31, 65, 13, ES_AUTOHSCROLL |
WS_BORDER | WS_TABSTOP
EDITTEXT ID_CONF_DETECT, 75, 53, 65, 13,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_CONF_FLOW, 75, 74, 65, 13, ES_AUTOHSCROLL
| WS_BORDER | WS_TABSTOP
EDITTEXT ID_CONF_MEASTEMP, 75, 95, 65, 13,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_CONF_PRESS, 75, 112, 65, 13,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
CONTROL "", IDOK, "BorBtn", BS_DEFPUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 57, 138, 37, 25
CONTROL "", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 113, 138, 37, 25
CONTROL "", -1, "BorShade", BSS_GROUP | BSS_LEFT |
WS_CHILD | WS_VISIBLE, 6, 4, 186, 126
RTEXT "Reactor Volume :", -1, 13, 13, 60, 8
RTEXT "Value of R :", -1, 31, 33, 42, 8
RTEXT "Detector Used :", -1, 19, 55, 54, 8
LTEXT "cm3", -1, 146, 13, 16, 8
LTEXT "J mol-1 K-1", -1, 146, 32, 37, 10
RTEXT "Measurement Temp :", -1, 23, 92, 50, 19
LTEXT "deg C", -1, 146, 96, 37, 10
RTEXT "Reactor Press :", -1, 13, 114, 60, 8
LTEXT "psi", -1, 146, 113, 37, 10
RTEXT "Flow Rate :", -1, 20, 76, 53, 8
LTEXT "cm3 s-1", -1, 146, 75, 37, 10
}

ID_KINET_MENU MENU
{
POPUP "&File"
 {
MENUITEM "&Open\tCTRL - O", CM_FILE_OPEN
MENUITEM "&Save\tCTRL - S", CM_FILE_SAVE
MENUITEM "Save&As", CM_FILE_SAVEAS
MENUITEM "&Close", CM_FILE_CLOSE
MENUITEM SEPARATOR
MENUITEM "&Import", CM_IMPORT
MENUITEM "&Export", CM_EXPORT
MENUITEM SEPARATOR
MENUITEM "E&xit\tALT - X", CM_EXIT
 }

POPUP "&Edit"
 {
MENUITEM "&Order", CM_EDIT_ORDER
 }

POPUP "&View"
 {
MENUITEM "&Kinetic  Data\tAlt - K", CM_VIEWKINETIC_DATA
POPUP "&Calibration"
  {
MENUITEM "&Reactant\tAlt - R", CM_VIEW_REACTANT
MENUITEM "&Product\tAlt - P", CM_VIEW_PRODUCT
  }
 }

POPUP "C&alculate"
 {
POPUP "&Calibration"
  {
MENUITEM "&Reactant", CM_CALC_CALREACT
MENUITEM "&Product", CM_CALC_CALPROD
  }

MENUITEM "&Amounts\tAlt - A", CM_CALC_AMOUNTS
MENUITEM "&Order\tAlt - O", CM_CALC_ORDER
MENUITEM "Activation &Energy", CM_CALC_ACTENERGY
 }

POPUP "&Data"
 {
```

```
MENUITEM "&New Pyrolysis Data Entry\tCTRL - K",
CM_DATA_NEWKIN
POPUP "New &Calibration Entry"
 {
MENUITEM "&Reactant\tCTRL - R", CM_DATA_NEWREACT
MENUITEM "&Product\tCTRL - P", CM_DATA_NEWPROD
 }

MENUITEM SEPARATOR
MENUITEM "&Edit Data\tCTRL - E", CM_DATA_EDIT
MENUITEM "&Delete\tDel", CM_DATA_DELETE
}

POPUP "&Config"
{
MENUITEM "&Basic\tCTRL - C", CM_CONFIG
}

MENUITEM "Debug &Info", CM_DEBUG_INFO
}

ID_PROD_DLG DIALOG 5, 46, 293, 113
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU
CLASS "BorDlg_Gray"
CAPTION "Product"
FONT 10, "Arial"
{
EDITTEXT ID_PROD_RETENT, 76, 48, 68, 12,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_PROD_PEAKAREA, 76, 63, 68, 12,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_PROD_NAME, 76, 16, 68, 12, ES_AUTOHSCROLL
| ES_NOHIDESEL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_PROD_FORM, 76, 32, 68, 12,
ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP
EDITTEXT ID_PROD_COEFFM, 218, 27, 53, 13
EDITTEXT ID_PROD_COEFFC, 218, 43, 53, 13
CONTROL "", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 102, 86, 33, 21
CONTROL "", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 158, 86, 33, 21
RTEXT "Compound Name :", -1, 13, 16, 59, 9, SS_RIGHT |
WS_GROUP
CTEXT "y = m x + c", -1, 166, 54, 42, 10, SS_CENTER |
WS_BORDER | WS_GROUP
RTEXT "Coeff m :", -1, 183, 29, 31, 9, SS_RIGHT | WS_GROUP
RTEXT "Coeff c :", -1, 183, 43, 31, 9, SS_RIGHT | WS_GROUP
RTEXT "Peak Area :", -1, 33, 64, 39, 8, SS_RIGHT | WS_GROUP
CONTROL "Detector Sensitivity", -1, "BorShade", BSS_GROUP |
BSS_CAPTION | BSS_LEFT | WS_CHILD | WS_VISIBLE, 149, 14,
130, 53
RTEXT "Chemical Formula", -1, 38, 29, 34, 18, SS_RIGHT |
WS_GROUP
CONTROL "Compound", -1, "BorShade", BSS_GROUP |
BSS_CAPTION | BSS_LEFT | WS_CHILD | WS_VISIBLE, 6, 3,
281, 75
RTEXT "Retention Time :", -1, 33, 49, 39, 8
}

ID_KINET_MENU ACCELERATORS
{
"^S", CM_FILE_SAVE, ASCII, NOINVERT
"x", CM_EXIT, ASCII, NOINVERT, ALT
"^C", CM_CONFIG, ASCII, NOINVERT
"^E", CM_DATA_EDIT, ASCII, NOINVERT
"^O", CM_FILE_OPEN, ASCII, NOINVERT
VK_DELETE, CM_DATA_DELETE, VIRTKEY, NOINVERT
"^D", CM_DEBUG_INFO, ASCII, NOINVERT
"r", CM_VIEW_REACTANT, ASCII, NOINVERT, ALT
"p", CM_VIEW_PRODUCT, ASCII, NOINVERT, ALT
"k", CM_VIEWKINETIC_DATA, ASCII, NOINVERT, ALT
"a", CM_CALC_AMOUNTS, ASCII, NOINVERT, ALT
"o", CM_CALC_ORDER, ASCII, NOINVERT, ALT
"^K", CM_DATA_NEWKIN, ASCII, NOINVERT
"^R", CM_DATA_NEWREACT, ASCII, NOINVERT
"^P", CM_DATA_NEWPROD, ASCII, NOINVERT
}


DLG_INPUT DIALOG 76, 61, 154, 64
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU
CLASS "BorDlg_Gray"
CAPTION "Delimated Text Information"
FONT 8, "MS Sans Serif"
{
EDITTEXT IDC_INPUT_EDIT, 51, 25, 51, 13, ES_CENTER |
WS_BORDER | WS_TABSTOP
DEFPUSHBUTTON "OK", IDOK, 52, 48, 50, 14
CTEXT "Enter the character to use to delimate text!", -1, 4, 10, 146, 10
}

ID_CAL_DLG DIALOG 39, 54, 190, 104
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU
CLASS "BorDlg_Gray"
CAPTION "Calibration Entry"
FONT 10, "Arial"
{
EDITTEXT ID_CAL_NAME, 65, 10, 42, 12, ES_AUTOHSCROLL |
WS_BORDER | WS_TABSTOP
EDITTEXT ID_CAL_PRESS, 65, 34, 42, 12, ES_AUTOHSCROLL |
WS_BORDER | WS_TABSTOP
EDITTEXT ID_CAL_TEMP, 65, 55, 42, 12, ES_AUTOHSCROLL |
WS_BORDER | WS_TABSTOP
EDITTEXT ID_CAL_PEAK, 65, 78, 42, 12, ES_AUTOHSCROLL |
WS_BORDER | WS_TABSTOP
CONTROL "", IDOK, "BorBtn", BS_DEFPUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 147, 25, 37, 25
CONTROL "", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 147, 60, 37, 25
CONTROL "", -1, "BorShade", BSS_RGROUP | BSS_LEFT |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 137, 17, 54, 69
RTEXT "Pressure Injected", -1, 6, 35, 56, 9
RTEXT "Temperature Injected", -1, 6, 52, 56, 19
RTEXT "Peak Area", -1, 26, 79, 36, 9
LTEXT "torr", -1, 112, 36, 15, 8
LTEXT "deg C", -1, 112, 57, 22, 8
LTEXT "uV s", -1, 112, 80, 15, 8
RTEXT "Compound name", -1, 6, 11, 56, 9
KINET1_ICON ICON "kinet1.ico"
```

Consts.cpp

```
// this file contains the constant data for the program
// it is held seperately to allow changes to be made without making a
// large amount of recompiling necessary


#include "data_def.h" // contains data constant defines
#include "kinet1b.h"  // contains extern declarations of all data here.
// included here to force type checking to take place and avoid hard to
solve errors
const char Version[] = "Kinetic Calculator For Windows v1.01";
const char ConfigTitleText[] = "Reactor Pressure (psi)\t" "Measured
Flow (cm3 s-1)\t"
"Measurement Temperature (C)\t" "Reactor vol (cm3)\t"
"Value of R used (J mol-1 K-1)\t" "Detector Used";
const int BufSize = 1024;  // size of buffer used in import function to
check format
const double DefReactorVol = 20.6;  // default reactor volume used at
startup
const double DefGasConstant = R;  // default gas constant
const char DefDetector[] = "FID";  // default detector type
// opensave dialog text for filters etc
const char ExpExt[] = ".TXT";  // default extension for import / export
files
const char OpenSaveFilter[] = "Kinetic files (*.kin)|*.kin|"
"All Files (*.*)|*.*|";
const char ImportExportFilter[] = "Text Files (*.txt)|*.txt|";
const char OpenSaveExt[] = ".KIN";  // default extension for data normal
files
// text for title in Kinetic data view.  See KinetData::Sprintf in
// Kineticls.cpp for order of output
const char KineticTitleText[] =    // 30 tabs
"Date Performed\t" "Mass Balance Ratio\t" "Trap Time (min)\t"          //
3 tabs
"Injection Press (torr)\t" "Injection temp (C)\t"  "Injection temp (K)\t"  // 3
tabs
"Amount Injected (mol)\t" "Reactor Temp (C)\t"  "Reactor Temp (K)\t"
// 3 tabs
"Reactant Name\t" "Reactant Formula\t" "Reactant Ret. Time (min)\t"
// 3 tabs
"Reactant Peak Area\t" "Reactant Sens Coeff M\t" "Reactant Sens
Coeff C\t" // 3 tabs
"Product Name\t" "Product Formula\t" "Product Ret. Time (min)\t"
// 3 tabs
"Product Peak Area\t" "Product Sens Coeff M\t" "Product Sens Coeff
C\t"   // 3 tabs
"React Amount\t"  "Product Amount\t"                                  // 2 tabs
"log(Reactant)\t" "log(Product)\t"                                    // 2 tabs
"Reaction Order\t"  "Adjusted Flow Rate\t"   "Tau\t"  "Rate Constant\t"
// 4 tabs
"1 / T \t" "ln k";                                    // 1 tab


// const int KineticNumTabs = 30;  this is a define in kinet1b.h now
// tab positions in characters. Used later to calculate coordinates for
actual tabs
const int KinetictabChar[KineticNumTabs] = { 25, 25, 25, 25, 25,
25, 25, 25, 25, 25,
25, 30, 25, 30, 30,
30, 20, 30, 30, 30,
30, 25, 25, 18, 18,
22, 30, 15, 20, 15 };
// text for Product/Reactant view title
const char ProdReactTitleText[] =        // 5 tabs
"Compound Name\t"  "Press Injected\t"   "Amount Injected\t"  "Injection
Temp (C)\t"
"Injection Temp (K)\t"   "Peak Area (uV s)";
// const int ProdReactNumTabs = 5;   this is a define in kinet1b.h now
// tab positions in characters. Used later to calculate coordinates for
actual tabs
const int ProdReacttabChar[ProdReactNumTabs] = { 30, 30, 30, 30,
30};

const char LinRegressTitle[] =   // 4 tabs
"Coeff M\t"  "Coeff C\t"   "R squared\t" "rms error\t"
"PerCentError in M";
const int LinRegrTitTab[5] = { 20, 20, 20, 20, 20};
//const int NumRegrTitTab = 5;  this is a define in kinet1b.h now


// title of arrhenius fields
const char TArrheniusTitle[] =        // 1 tab
"Activation Energy\t"  "ln(Arrhenius Param)";
const int TArrheniusTitTab[3] = { 25, 25, 35 };
//const int NumTArrheniusTitTab = 2;    this is a define in kinet1b.h
now
```

Extr.cpp

```
// this file contains macro calls to write functions. not including those
// macros that are part of OWL. Only those macros that were written
by me
#include "kinet1a.h"
#include "macextr.h"
// use macro to write extraction functions
// names are then Extract_function_name
// calibration data extract
Extract_write(Cal, TCalibrationData, amount, peakarea)
// kinetic data extract for order plots
Extract_write(Kin_Order, TKinData, logReact, logProd)
// kinetic data extract for activation energy plots
Extract_write(Act_Engy, TKinData, InverseTemp, lnRateConst)
```

Genfunct.cpp

```
#include <strstrea.h>
#include <cstring.h>
#include <string.h>
#include <math.h>
#include "genfunct.h"

/*_____________________________________________
// string search functions
_____________________________________________*/
int CharReplace(char* s, const char* find, const char r)
 {
char replace[2];
replace[0] = r;
replace[1] = '\0';
return CharReplace(s, find, replace); // calls other version of itself
 }
int CharReplace(char* s, const char* find, const char* r)
 {
 string str(s);     // target string
 string replace®;  // replacement string
size_t pos = 0; // position marker
int FLen = strlen(find); // length of the string to be replaced
int NumReplace = 0; // number of replacements made
if(FLen > str.length() ) return -1; // not room to find the find string in s
pos = str.find( (string) find, pos ); // find first occurence
if(pos != NPOS)  // i.e. the string was found
            {
str = str.replace(pos, FLen, replace);
NumReplace++;
            }
while( pos != NPOS )
 {
pos = str.find( (string) find, pos + FLen ); // find next occurence
if(pos != NPOS)  // i.e. the string was found
            {
str = str.replace(pos, FLen, replace);
NumReplace++;
            }
 }
return NumReplace;
 }
int CharCount(char* s, int ch)
// counts occurences of character ch in string s. returns number found
or -1 on error
 {
int y=0;
BOOL Found = TRUE;
char* p = s;
if( ch == 0) return -1; // searching for null terminators is an error
// as the length of the string to be searched is not passed to this
function
while(Found == TRUE)
 {
if(p = strchr(p+1, ch)) y++;
else Found = FALSE;
 }
return y;
 }
/*_____________________________________________
STRING MANIPULATION functions
_____________________________________________*/
//truncates at first non-digit (excluding '.')
// returns TRUE if there is any string remaining
BOOL NumTruncate(char* input)
 {
long len = strlen(input);
long i = 0; // index
while( !(i > len ) )  // until the end of the string
 {
if( !isdigit(input[i]) && (input[i] != '.') ) // not a digit or a '.'
break;  // exit loop
```

```
i++;
 }
input[i] = '\0'; // truncate string
if(strlen(input)) return TRUE;  // string still has some length
else return FALSE;
}
/*_____________________________________________
stats functions
_____________________________________________*/

LinRegress LinearRegress(double* X, double* Y, long NumPoints)
{
// performs linear regression using the arrays of doubles
// assumes NumPoints data points.  Possibly causes errors if
// NumPoints is wrong.
// returns values of coefficients and errors in a LinRegress structure
if( (!X || !Y) || ( NumPoints <= 0 ) )// data not passed properly
 {
LinRegress l;
l.CorrCoeff = 50;  // impossible value for R
return l;
 }
double temp1, temp2;    // for temporary storage of data for error
checks
// deviation of each point from the calculated point
double *deltas = new double[NumPoints];
// deviation of each x from the mean x
double *DevMeanX = new double[NumPoints];
LinRegress l;
double SumXi, SumYi, SumXi_sqr, SumYi_sqr, SumXiYi;  // values
needed later
double SumDeltas, meanX, SumAdjDelta;
SumDeltas = SumXi  = SumYi  =  SumXi_sqr = SumYi_sqr =
SumXiYi = 0;
meanX = SumAdjDelta = 0;
// calculate sums above
for( long i=0; i < NumPoints; i++)
 {
SumXi += *(X+i);
SumYi += Y[i];
SumXi_sqr +=  X[i] * X[i];
SumYi_sqr +=  Y[i] * Y[i];
SumXiYi  +=  X[i] * Y[i];
 }
meanX = SumXi / NumPoints;
for(i=0; i < NumPoints; i++)
 {
DevMeanX[i] = X[i] - meanX;
 }
// calculate m
temp1 = ((SumXi * SumXi) - (NumPoints * SumXi_sqr));
if(temp1)  // to stop divide by zero errors
 {
 l.m =  ((SumXi * SumYi) - (NumPoints * SumXiYi))  // this
                          /                        // divided by
temp1; // this
 }
else l.m = 0;
// calculate intercept
temp1 = ((SumXi * SumXi) - (NumPoints * SumXi_sqr));
if(temp1)   // to stop divide by zero errors
 {
l.c =  ((SumXiYi * SumXi) - (SumXi_sqr * SumYi)) // this
                          /                        // divided by
                        temp1;   // this
 }
else l.c = 0;
// calculate deviations and their sum
for(i=0; i < NumPoints; i++)
 {
deltas[i] = Y[i] - l.m * X[i] - l.c;  // ie the real point - calulated point
SumDeltas += deltas[i] * deltas[i]; //* deltas[i];
 }
// calculate rms
l.rms = sqrt((SumDeltas / NumPoints));
// calculate correlation coefficient
temp1 = sqrt( ((SumYi * SumYi) - (NumPoints * SumYi_sqr))   //
square root of these
                                                            *
((SumXi * SumXi) - (NumPoints * SumXi_sqr)) );
if(temp1)    // as long as temp1 is not zero ( to stop divide by zero
errors)
 {
 l.CorrCoeff = ((SumXi * SumYi) - (NumPoints * SumXiYi))     // this
                                                             /
// divided by
                                                          temp1;
// two lines
 }
else l.CorrCoeff = 200; // to show when output that no real value was
calculated
l.CorrCoeff_sqr = square( l.CorrCoeff );
// calculate % -age error in m
// need Sumof( [(Xi - meanX)delta] ^2 )
for(i=0; i < NumPoints; i++)
 {
SumAdjDelta += square(  DevMeanX[i] * deltas[i]);
 }
// calculate error in coefficient m
temp1 = ((NumPoints * SumXiYi) - (SumXi * SumYi));
if(temp1)  // stop divide by zero errors
 {
l.PerCentErrorM = 100 *
                                    (
(NumPoints * (sqrt(SumAdjDelta)) )
                                    /
temp1
                                    );
          }
else  l.PerCentErrorM = 100;
delete DevMeanX;
delete deltas;
return l;
}
double square(double d)
{
if( (d < (1.7e308 / 2)) && (d > (-1.7e308 / 2) ) )
 {
return d * d;
 }
else return -1;
}
/*_____________________________________________
Functions for Data objects needed
_____________________________________________*/
LinRegress::LinRegress()
{ reset(); }
void LinRegress::reset()
{ n = m = c = CorrCoeff = CorrCoeff_sqr = rms = PerCentErrorM = 0;
}
void LinRegress::Sprintf(char* str, int Len)
{
ostrstream out(str, Len);
out << "\t" << m << "\t" << c << "\t" << CorrCoeff_sqr << "\t" << rms
<< "\t" << PerCentErrorM;
out.put('\0');  // end string
}
```

Kinet1a.cpp

```
#include "kinet1a.h"
#include "kinet1b.h"
#include "genfunct.h"
/*-------------------------------------
Dialog box
-------------------------------------*/
TInputBox::~TInputBox()
{
delete InputBox;
}
TProdDialog::~TProdDialog()
{
delete CompName;
delete ChemForm;
delete RetentTime;
delete PeakArea;
delete CoeffC;
delete CoeffM;
}
TCalDialog::~TCalDialog()
{
delete Name;
delete PressInj;
delete TempInj;
delete PeakArea;
}
TConfigDialog::~TConfigDialog()
{
delete Flow;
delete PRESS;
delete MeasTemp;
delete ConstR;
delete Detect;
delete ReactVol;
}
TReactDialog::~TReactDialog()
{
delete CompName;
delete ChemForm;
delete DatePerformed;
delete RetentTime;
delete PeakArea;
delete TrapTime;
delete ReactorTemp;
delete CoeffC;
delete CoeffM;
delete InjPress;
delete InjTemp;
}
TReactDialog::TReactDialog(TWindow* parent, TResId resId)
: TWindow(parent), TDialog(parent, resId)
{
CompName = new TEdit(this, ID_REACT_NAME, MaxEditLen);
ChemForm  = new TEdit(this, ID_REACT_CHEMFORM,
MaxEditLen);
DatePerformed = new TEdit(this, ID_REACT_DATE, MaxEditLen);
RetentTime = new TEdit(this, ID_REACT_RETENT, MaxEditLen);
PeakArea  = new TEdit(this, ID_REACT_PEAKAREA, MaxEditLen);
TrapTime = new TEdit(this, ID_REACT_TRAP, MaxEditLen);
ReactorTemp = new TEdit(this, ID_REACT_REACTTEMP,
MaxEditLen);
CoeffC = new TEdit(this, ID_REACT_COEFFC, MaxEditLen);
CoeffM  = new TEdit(this, ID_REACT_COEFFM, MaxEditLen);
InjPress = new TEdit(this, ID_REACT_INJPRESS, MaxEditLen);
InjTemp  = new TEdit(this, ID_REACT_TEMPINJ, MaxEditLen);
// assign the address of the dialog transfer buffer to member
TransferBuffer
TransferBuffer = (void far*)&(((TMainWindow*)Parent)->ReactTrans);
 }
TInputBox::TInputBox(TWindow* parent, TResId resId)
: TWindow(parent), TDialog(parent, resId)
{
InputBox =  new TEdit(this, IDC_INPUT_EDIT, MaxEditLen);
// assign the address of the dialog transfer buffer to member
TransferBuffer
TransferBuffer = (void far*)&(((TMainWindow*)Parent)->InputTrans);
}
TCalDialog::TCalDialog(TWindow* parent, TResId resId)
:  TWindow(parent), TDialog(parent, resId)
{
Name =    new TEdit(this, ID_CAL_NAME, MaxEditLen);
PressInj = new TEdit(this, ID_CAL_PRESS, MaxEditLen);
TempInj = new TEdit(this, ID_CAL_TEMP, MaxEditLen);
PeakArea = new TEdit(this, ID_CAL_PEAK, MaxEditLen);
// assign the address of the dialog transfer buffer to member
TransferBuffer
TransferBuffer = (void far*)&(((TMainWindow*)Parent)->CalTrans);
}
TConfigDialog::TConfigDialog(TWindow* parent, TResId resId)
: TWindow(parent), TDialog(parent, resId)
{
Flow  = new TEdit(this, ID_CONF_FLOW, MaxEditLen);
PRESS = new TEdit(this, ID_CONF_PRESS, MaxEditLen);
MeasTemp  = new TEdit(this, ID_CONF_MEASTEMP,
MaxEditLen);
ConstR = new TEdit(this, ID_CONF_R, MaxEditLen);
Detect = new TEdit(this, ID_CONF_DETECT, MaxEditLen);
ReactVol = new TEdit(this, ID_CONF_VOL, MaxEditLen);
// assign the address of the dialog transfer buffer to member
TransferBuffer
TransferBuffer = (void far*)&(((TMainWindow*)Parent)-
>ConfigTrans);
}
TProdDialog::TProdDialog(TWindow* parent, TResId resId)
: TWindow(parent), TDialog(parent, resId)
{
CompName = new TEdit(this, ID_PROD_NAME, MaxEditLen);
ChemForm = new TEdit(this, ID_PROD_FORM, MaxEditLen);
RetentTime =  new TEdit(this,ID_PROD_RETENT , MaxEditLen);
PeakArea = new TEdit(this, ID_PROD_PEAKAREA, MaxEditLen);
CoeffC  = new TEdit(this, ID_PROD_COEFFC, MaxEditLen);
CoeffM =  new TEdit(this, ID_PROD_COEFFM, MaxEditLen);
// assign the address of the dialog transfer buffer to member
TransferBuffer
TransferBuffer = (void far*)&(((TMainWindow*)Parent)->ProdTrans);
}
/*-------------------------------------------------
TMainWindow calculation functions
-------------------------------------------------*/
/* ------EDIT MENU functions----------*/
void TMainWindow::EditOrder()
{
if( KinData->IsEmpty() ) return;  // no data to work with
long NumCalRecs = KinData->GetItemsInContainer();
BOOL ClearSelected = FALSE;
char input[MaxEditLen] = "";
if(TInputDialog(this, "Reaction Order Change",
"Enter the new reaction order",
input, sizeof(input) ).Execute() == IDOK)
 {
if( NumTruncate(input) ) // NumTruncate truncates a string at the first
non-digit
// (it ignores '.' ) returns TRUE if a string remains after truncation
            {
if(KinSelected->IsEmpty() ) // no selected records, list all records
            {
ClearSelected = TRUE;
for(long j = 0; j < NumCalRecs; j++)
            {
KinSelected->Add(j);
            }
            }
TLongArrayIterator i(*KinSelected);
while(i)
            {
```

```
double num = atof(input);
((*KinData)[i++]->data() ).Order = num;
                }
              }
else // there was not enough number in the string to obtain a value
MessageBox("You must enter a number",
"Order Change Error", MB_OK);
 }
if(ClearSelected) KinSelected->Flush(); // clear selected list if
necessary
Invalidate(FALSE);
UpdateWindow();
}

/* ---------------------------------------------------
CALCULATION functions
---------------------------------------------------*/
void TMainWindow::CalcOrder()
 {
double *logReact, *logProd;
long NumCalRecs = KinData->GetItemsInContainer();
BOOL ClearSelected = FALSE;
if(KinSelected->IsEmpty() ) // no selected records, list all records
 {
ClearSelected = TRUE;
for(long j = 0; j < NumCalRecs; j++)
            {
KinSelected->Add(j);
            }
 }
long NumPoints = KinSelected->GetItemsInContainer(); // find number
of records to work with
if(NumPoints <= 2)  // 3 points are the minimum for regression that
allows
// calculation of errors and R etc
 {
MessageBox("You must select more than two records for the "
"sensitivity to be calculated",
"Regression Error", MB_OK);
return; // exit function
 }
 if(! (logReact = new double[NumPoints]) ) return;   // failure of memory
allocation
 if(! (logProd = new double[NumPoints]) ) return;
  // build arrays of doubles for regression
 Extract_Kin_Order( logReact, logProd, *KinData, *KinSelected );
 KinRegress = LinearRegress(logReact, logProd, NumPoints);
 TLongArrayIterator i(*KinSelected);
 while(i)    // set all orders to calculated value
  {
 (((*KinData)[i++])->data()).Order = KinRegress.m;
  }
if(ClearSelected) KinSelected->Flush();
delete logReact;
delete logProd;
Invalidate(FALSE);
UpdateWindow();
}
void TMainWindow::CalcAmounts()
 {
// this calculates the amounts for the kinetic data records
TKinDataIterator i(*KinData);
while(i)
 {
(i++)->CalcAmounts();
 }
Modified(TRUE);
UpdateScrollData();
Invalidate(FALSE);
UpdateWindow();
}
void TMainWindow::CalcActEngy()
 {
if(KinData->IsEmpty() ) return;  // no data to work on
if(CalcActivation_Energy(*KinData, *KinSelected) == 2)
MessageBox("You must select more than two records for the "
"sensitivity to be calculated"
"Regression Error", MB_OK);
Invalidate(FALSE);
UpdateWindow();
}
int TMainWindow::CalcActivation_Energy(TKinData& data,
TLongArray& selected)
 {
double *InvTemp, *lnRateConst; // for arrays of double data
BOOL ClearSelected = FALSE;
long NumCalRecs = data.GetItemsInContainer();
if(selected.IsEmpty() ) // no selected records, list all records
 {
ClearSelected = TRUE;
for(long i = 0; i < NumCalRecs; i++)
            {
selected.Add(i);
            }
 }
long NumPoints = selected.GetItemsInContainer(); // find number of
records to work with
if(NumPoints <= 2) return 2;  // 3 points are the minimum for regression
that allows
// calculation of errors and R etc
if(! (InvTemp = new double[NumPoints]) ) return 0;
if(! (lnRateConst = new double[NumPoints]) ) return 0;
// build arrays of doubles for regression
Extract_Act_Engy( InvTemp, lnRateConst, data, selected );
// KinArr is a TArrhenius object
KinArr = KinRegress = LinearRegress(InvTemp, lnRateConst,
NumPoints);
if(ClearSelected) KinSelected->Flush();
delete InvTemp;
delete lnRateConst;
return 1; // successful
}
void TMainWindow::HandleCalCalcReact()
 {
if(ReactCalData.data().IsEmpty() ) return;  // no data to work on
if(CalcCal(ReactCalData, *ReactSelected) == 2)
MessageBox("You must select more than two records for the "
"sensitivity to be calculated"
"Regression Error", MB_OK);
Invalidate(FALSE);
UpdateWindow();
}
void TMainWindow::HandleCalCalcProd()
 {
if(ProdCalData.data().IsEmpty() ) return;  // no data to work on
if(CalcCal(ProdCalData, *ProdSelected) == 2)
MessageBox("You must select more than two records for the "
"sensitivity to be calculated"
"Regression Error", MB_OK);
Invalidate(FALSE);
UpdateWindow();
}
int TMainWindow::CalcCal(TProductReactantCal& cal, TLongArray&
selected)
 {
double *amounts, *peakareas;
BOOL ClearSelected = FALSE;
TCalibrationData&  d = cal.data();
long NumCalRecs = d.GetItemsInContainer();
if(selected.IsEmpty() ) // no selected records, list all records
 {
ClearSelected = TRUE;
for(long i = 0; i < NumCalRecs; i++)
            {
selected.Add(i);
            }
 }
```

```
long NumPoints = selected.GetItemsInContainer(); // find number of
records to work with
if(NumPoints <= 2) return 2; // 3 points are the minimum for regression
that allows
// calculation of errors and R etc
if(! (amounts = new double[NumPoints]) ) return 0;
if(! (peakareas = new double[NumPoints]) ) return 0;
// build arrays of doubles for regression
Extract_Cal( amounts, peakareas, d, selected );
cal.regress() = LinearRegress(amounts, peakareas, NumPoints);
if(ClearSelected) selected.Flush();
delete amounts;
delete peakareas;
return 1;
}
/*_____________________________________________
General functions
_____________________________________________*/
// operator overloading for TConfigData
ofstream& operator << (ofstream& os, TConfigData& data)    // for text
output
{
int oldPrec = os.precision(6);
os << Delimator << data.ReactorVol << Delimator;  // use file's default
delimator
os.precision(6);
os << data.GasConstant << Delimator;
os.precision(6);
os << data.Detector << Delimator;
os.precision(oldPrec);
return os;
}
ifstream& operator >> (ifstream& is, TConfigData& data) // for text input
{
char vol[30];
char G[30];
is.ignore(2, Delimator); // ignore first delimator in line
is.getline(vol, 29, Delimator);   // use file's default delimator
is.getline(G, 29, Delimator);
is.getline(data.Detector, 29, Delimator);
data.ReactorVol = atof(vol);
data.GasConstant = atof(G);
return is;
}
/*_____________________________________________
TProductReactantCal functions
_____________________________________________*/
void TProductReactantCal::Clear()
{
Regress.reset();
if( !Data->IsEmpty() ) Data->Flush(TShouldDelete::Delete);
}
ofstream& TProductReactantCal::Save(ofstream& os, char Delimator,
ios::open_mode mode)
{
TCalDataIterator i("Data);
switch (mode)
 {
case ios::binary :
            {
os.write( (char*)&Regress, sizeof(LinRegress) );
break;
            }
case ios::out :
            {
char title[MaxEditLen];
char tmp[3];
tmp[0] = Delimator;
tmp[1] = '\0';
strcpy(title, LinRegressTitle);
            CharReplace(title, "\t", tmp);          // format title text with
delimator
            os << title << Delimator << "\n"    // write title of regression
fields
                        << Delimator << Regress.m << Delimator <<
Regress.c  // write regression fields
                        << Delimator << Regress.CorrCoeff
                        << Delimator << Regress.rms << Delimator
                        << Regress.PerCentErrorM << "\n";
            strcpy(title, ProdReactTitleText);       // format title text with
delimator
CharReplace(title, "\t", tmp);
os << Delimator << title << Delimator << "\n";  // write title text
break;
            }
default :   // not the correct mode
            {
return os;
            }
 }  // end of switch
while(i)
 {
if(mode == ios::out) os << Delimator;
(i++)->Save(os, Delimator, mode);
if(mode == ios::out)   os << "\n";
 }
return os;
}  // end of ofstream& TProductReactantCal::Save(... ... ...)
ifstream& TProductReactantCal::Load(ifstream& is, char Delimator, int
numrecs, ios::open_mode mode)
{
TCalData* tmp;
if( !Data->IsEmpty() ) Data->Flush( TShouldDelete::Delete);
if(numrecs)
 {
switch (mode)
            {
case ios::binary :
            {
is.read( (char*)&Regress, sizeof(LinRegress) );
break;
            }
case ios::in  :
            {
break;
            }
default :   // not the correct mode
            {
return is;
            }
}  // end of switch
for(int i=0; i < numrecs; i++)
            {
Data->Add( tmp = new TCalData );
tmp->Load(is, Delimator, mode);
            }
 }
return is;
}   // end  of  ifstream& TProductReactantCal::Load(... ... ...)
/*_____________________________________________
TArrhenius functions
_____________________________________________*/
TArrhenius::TArrhenius()
{ reset(); }
TArrhenius::TArrhenius(double Ea, double A)
{
ActEngy = Ea;
ArrParam = A;
}
void TArrhenius::reset()
{
ActEngy = ArrParam = 0;
}
void TArrhenius::Sprintf(char* str, int Len)
{
ostrstream out(str, Len);
out << ActEngy << "\t" << ArrParam;
out.put('\0');
```

```
}
ofstream& TArrhenius::Save(ofstream& os, ios::open_mode mode,
char Delimator)
{
switch (mode)
 {
case (ios::binary) :
            {
os.write( (char*) &ActEngy, sizeof(double) );
os.write( (char*) &ArrParam, sizeof(double) );
break;
            }
case (ios::out) :
            {
os << ActEngy << Delimator << ArrParam;
break;
            }
 }
return os;
}
ifstream& TArrhenius::Load(ifstream& is, ios::open_mode mode, char
Delimator)
{
switch (mode)
 {
case (ios::binary) :
            {
is.read( (char*) &ActEngy, sizeof(double) );
is.read( (char*) &ArrParam, sizeof(double) );
break;
            }
case (ios::in) :
            {
char temp[MaxEditLen];
is.getline(temp, MaxEditLen-1, Delimator);
ActEngy = atof(temp);
is.getline(temp, MaxEditLen-1, Delimator);
ArrParam = atof(temp);
break;
            }
 }
return is;
}
TArrhenius& TArrhenius::operator =(LinRegress& l)
{
ActEngy = -Lm * R;
ArrParam = Lc;
return *this;
}
/*_____________________________________________________
General functions for calculation routines
_____________________________________________________*/
```

Kinet1ts.cpp

```
#include "kinet1.h"
#include "kinetds.h"
/*_____________________________________________________
TConfigData
_____________________________________________________*/
TConfigData::TConfigData()
            {
ReactorPress = FlowRate = MeasTemp = ReactorVol = GasConstant
= 0;
strcpy(Detector, "");
            }
inline TConfigData& TConfigData::operator=(double num)
{
ReactorPress = num;
FlowRate = num;
MeasTemp = num;
ReactorVol = num;
GasConstant = num;
sprintf(Detector, "%g", num);
return *this;
}
void TConfigData::Set(TConfigTransferBuffer& c)
            {
ReactorPress = atof(c.ReactorPress);
FlowRate = atof(c.FlowRate);
MeasTemp = atof(c.MeasTemp);
ReactorVol =atof(c.ReactVol);
GasConstant = atof(c.ConstR);
strcpy(Detector, c.Detect);
            }
ofstream& TConfigData::Save(ofstream& os, char Delimator,
ios::open_mode mode)
{
switch (mode)
 {
case (ios::binary) :
            {
os.write( (char*) this, sizeof(TConfigData) );
break;
            }
case (ios::out) :
            {
os << ReactorPress << Delimator << FlowRate << Delimator
<< MeasTemp << Delimator << ReactorVol << Delimator
<< GasConstant << Delimator << Detector;
break;
            }
 }
return os;
}
long TConfigData::CheckData()  // checks for the presence of the data
{
long Missing = 0;
if(!ReactorPress) Missing |= REACTORPRESS;
if(!FlowRate) Missing |= FLOW;
if(!MeasTemp) Missing |= MEASTEMP;
if(!ReactorVol) Missing |= REACTORVOL;
if(!GasConstant) Missing |= GASCONST;
if(Detector[0] == '\0') Missing |= DETECTOR;

return Missing;
}
/*************************************************************
Transfer buffers
*************************************************************/
void TCalTransBuffer::Set(TCalDataStruct& b)
{
strcpy(Name, b.Name);
sprintf(PressInj, "%g", b.pressinj);
```

```
sprintf(TempInj, "%g", b.tempinjC);
sprintf(PeakArea, "%g", b.peakarea);
 }
void TProdTransferBuffer::Set(TKinDataStruct& k)
 {
strcpy(CompName, k.ProdName);
strcpy(ChemForm, k.ProdChemForm);
sprintf(RetentTime,"%g", k.ProdRetentTime);
sprintf(PeakArea,"%g", k.ProdPeakArea);
sprintf(CoeffC, "%g",  k.ProdCoeffC);
sprintf(CoeffM, "%g",  k.ProdCoeffM);
 }
void TReactTransferBuffer::Set(TKinDataStruct& k)
 {
strcpy(CompName,  k.ReactName);
strcpy(ChemForm,  k.ReactChemForm);
strcpy(DatePerformed, k.DatePerformed);
sprintf(RetentTime, "%g", k.ReactRetentTime);
sprintf(PeakArea, "%g", k.ReactPeakArea);
sprintf(TrapTime, "%g", k.TrapTime);
sprintf(ReactorTemp, "%g", k.ReactorTempC);
sprintf(CoeffC, "%g", k.ReactCoeffC);
sprintf(CoeffM, "%g", k.ReactCoeffM);
sprintf(InjPress, "%g", k.InjPress);
sprintf(InjTemp, "%g", k.InjTempC);
 }
/*_____________________________________________
TConfigTransferBuffer
_____________________________________________*/
void TConfigTransferBuffer::Set(TConfigData& c)
            {
sprintf(FlowRate, "%g", c.FlowRate);
sprintf(ReactorPress, "%g", c.ReactorPress);
sprintf(MeasTemp, "%g", c.MeasTemp);
sprintf(ConstR, "%g", c.GasConstant);
sprintf(ReactVol, "%g", c.ReactorVol);
strcpy(Detect, c.Detector);
            }
```

Kinetcls.cpp

```
#include <strstrea.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <cstring.h>
#if !defined __kinetcls_h__
#include "kinetcls.h"
#endif
/*************************************************
FileReadErr  member functions
*************************************************/
FileReadErr::~FileReadErr()
 {
if(str)  delete str;
 }
/*************************************************
TCalDataStruct  member functions
*************************************************/
void TCalDataStruct::Set(TCalTransBuffer& b)
 {
strcpy(Name, b.Name);
pressinj = atof(b.PressInj);
tempinjC = atof(b.TempInj);
tempinjK = tempinjC + CinK;  // conversion from C to K
peakarea = atof(b.PeakArea);
amount  = (pressinj * TORRtoPA) * SAMPLELOOP / ( R * tempinjK);
 }
void TCalDataStruct::Sprintf(ostrstream& stream, FIELDS f)// prints
set fields into stream
 {
switch (f)
 {
case ALL:
            {
stream << Name << "\t" << pressinj << "\t"  << amount << "\t" <<
tempinjC << "\t";
stream << tempinjK << "\t" << peakarea;
stream.put('\0');
            }
default :  { return; }
}  // end of switch
} // end ofTCalDataStruct::Sprintf(... )

ofstream& TCalDataStruct::Save(ofstream& os, char Delimator,
ios::open_mode mode)
 {
switch (mode)
 {
case ios::binary:
                        {
unsigned char n;  // to save string length to file
os.put( (n = ((unsigned char)strlen(Name)) ));
if(n) os << Name;
os.write( (char*)&pressinj, sizeof(double) );
os.write( (char*)&amount, sizeof(double) );
os.write( (char*)&tempinjC, sizeof(double) );
os.write( (char*)&tempinjK, sizeof(double) );
os.write( (char*)&peakarea, sizeof(double) );
return os;
                        }
case ios::out :
                        {
os << Name << Delimator << pressinj << Delimator  << amount <<
Delimator
<< tempinjC << Delimator << tempinjK << Delimator << peakarea;
return os;
                        }
default:   // if not correct mode
                        {
return os;      // exit
                        }
```

```
} // end of switch
} // end of TCalDataStruct::Save(...)
ifstream& TCalDataStruct::Load(ifstream& is, char Delimator,
ios::open_mode mode)
 {
switch (mode)
 {
case ios::binary:
                        {
unsigned char n; // to save string length to file
    is.read( (char*)&n, sizeof(unsigned char) ); // read in number of
bytes in next string
                        if( (n <= MAX) && (n) )   // less than max
length and greater than 0
                        {
is.read(Name, n); // read in string if not too long
                        }
else
                        {
if(n) throw(string( "Calibration data load error" )); // exit function if string
is too long
                        }
Name[n] = '\0';
is.read( (char*)&pressinj, sizeof(double) );
is.read( (char*)&amount, sizeof(double) );
is.read( (char*)&tempinjC, sizeof(double) );
is.read( (char*)&tempinjK, sizeof(double) );
is.read( (char*)&peakarea, sizeof(double) );
return is;
                        }
case ios::out :
                        {
char temp[MAX+1];
int max = MAX+1;

is.ignore(3, Delimator); // ignore first delimator in line
is.getline(Name, max, Delimator);
is.getline(temp, max, Delimator);
pressinj = atof(temp);
is.getline(temp, max, Delimator);
amount = atof(temp);
is.getline(temp, max, Delimator);
tempinjC = atof(temp);
is.getline(temp, max, Delimator);
tempinjK = atof(temp);
is.getline(temp, max, Delimator);
peakarea = atof(temp);
return is;
                        }
default:   // if not correct mode
                        {
return is;      // exit
                        }
} // end of switch
} // end of TCalDataStruct::Load( .. .. .. )
/*************************************************************
TCalData  member functions
*************************************************************/
TCalData::TCalData(TCalTransBuffer& b)
          {
Data.Set(b);
          }
void TCalData::Set(TCalTransBuffer& b)
 {
Data.Set(b);
 }
void TCalData::Sprintf(char* str, int sz , FIELDS f)// prints set fields into
str
 {
switch (f)
  {
case (ALL) :
          {
ostrstream stream(str, sz);
Data.Sprintf(stream, f);
break;
          }
case (REACT) : { break; }
case (PROD) : { break; }
default : { }
 }
}
ofstream& TCalData::Save(ofstream& os, char Delimator,
ios::open_mode mode)
 {
switch (mode)
 {
case ios::binary:
                        {
return Data.Save( os, Delimator, mode);
                        }
case ios::out :
 {
return Data.Save( os, Delimator, mode);
                        }
default:   // if not correct mode
                        {
return os;      // exit
                        }
} // end of switch
} // end of TCalData::Save( .. .. .. )
ifstream& TCalData::Load(ifstream& is, char Delimator,
ios::open_mode mode)
 {
switch (mode)
 {
case ios::binary:
                        {
return Data.Load( is, Delimator, mode);
                        }
case ios::out :
                        {
return Data.Load( is, Delimator, mode);
                        }
default:   // if not correct mode
                        {
return is;      // exit
                        }
} // end of switch
} // end of TCalData::Load( .. .. .. )
/*************************************************************
TKinDataStruct member functions
*************************************************************/
void TKinDataStruct::Reset()
 {
// one-off experimental data
strcpy(DatePerformed, "");
 InjPress = 0;   // torr
          InjTempC = 0;    // C
          InjTempK = 0;    // K
 AmountInj = 0;  // mol
 TrapTime = 0;  // min
 MassBalance = 0;    // ratio of amount in / amount out
// product data
ProdRetentTime = 0; // min
 ProdPeakArea = 0;  // uV s
 ProdCoeffM = 0;    // approx 1E+11
ProdCoeffC = 0; // usually 0
strcpy(ProdName, "");
strcpy(ProdChemForm, "");
// Reactant Data
 ReactRetentTime = 0;  // min
 ReactPeakArea = 0;    // uV s
 ReactCoeffM = 0;      // approx 1E+11
ReactCoeffC = 0; // usually 0
strcpy(ReactName, "");
strcpy(ReactChemForm, "");
// Non - constant Reactor Data
```

```
 AdjustedFlowRate = 0;   // cm3 s-1
 ReactorTempC = 0;       // C
 ReactorTempK = 0;       // K
// Results calculated
 RateConstant = 0;  // units depending on order
 ReactAmount = 0;  // mol calc from Peak areas and sensitivities
 ProdAmount = 0;  // mol calc from Peak areas and sensitivities
 Tau = 0;        // residence time of reactor / s
 Order = 1;      // order of the reaction
 logReact = 0;    // log of the amount of reactant detected for order plots
 logProd = 0;    // log of the amount of product detected for order plots
 lnRateConst = 0;  // natural log of rate constant
 InverseTemp = 0;       // 1/T  for plot against lnRateConst
 }
void TKinDataStruct::Set(TProdTransferBuffer& Prod)
 {
ProdRetentTime = atof(Prod.RetentTime);
ProdPeakArea = atof(Prod.PeakArea);
ProdCoeffM = atof(Prod.CoeffM);
ProdCoeffC = atof(Prod.CoeffC); // usually 0
strcpy(ProdName, Prod.CompName);
strcpy(ProdChemForm, Prod.ChemForm);
 }
void TKinDataStruct::Set(TReactTransferBuffer&  React)
 {
strcpy(DatePerformed, React.DatePerformed);
InjPress = atof(React.InjPress);
InjTempC = atof(React.InjTemp);
ReactPeakArea = atof(React.PeakArea);
ReactCoeffM = atof(React.CoeffM);
ReactCoeffC = atof(React.CoeffC); // usually 0
strcpy(ReactName, React.CompName);
strcpy(ReactChemForm, React.ChemForm);
ReactorTempC = atof(React.ReactorTemp);
TrapTime = atof(React.TrapTime);
ReactRetentTime = atof(React.RetentTime);
 }
ofstream& TKinDataStruct::Save(ofstream& os, char Delimator,
ios::open_mode mode)
 {
unsigned char n = 0;
if( mode == ios::binary)
  {
os.put(n = (unsigned char)strlen(DatePerformed));
if(n) os << DatePerformed;
os.write( (char*)&InjPress, sizeof(double) );
os.write( (char*)&InjTempC, sizeof(double) );
os.write( (char*)&InjTempK, sizeof(double) );
os.write( (char*)&AmountInj, sizeof(double) );
os.write( (char*)&TrapTime, sizeof(float) );
os.write( (char*)&MassBalance, sizeof(double) );
os.write( (char*)&ProdRetentTime, sizeof(double) );
os.write( (char*)&ProdPeakArea, sizeof(double) );
os.write( (char*)&ProdCoeffM, sizeof(double) );
os.write( (char*)&ProdCoeffC, sizeof(double) );
os.put(n = (unsigned char)strlen(ProdName));
if(n) os << ProdName;
os.put(n = (unsigned char)strlen(ProdChemForm));
if(n) os << ProdChemForm;
os.write( (char*)&ReactRetentTime, sizeof(double) );
os.write( (char*)&ReactPeakArea, sizeof(double) );
os.write( (char*)&ReactCoeffM, sizeof(double) );
os.write( (char*)&ReactCoeffC, sizeof(double) );
os.put(n = (unsigned char)strlen(ReactName));
if(n) os << ReactName;
os.put(n = (unsigned char)strlen(ReactChemForm));
if(n) os << ReactChemForm;
os.write( (char*)&AdjustedFlowRate, sizeof(double) );
os.write( (char*)&ReactorTempC, sizeof(double) );
os.write( (char*)&ReactorTempK, sizeof(double) );
os.write( (char*)&RateConstant, sizeof(double) );
os.write( (char*)&ReactAmount, sizeof(double) );
os.write( (char*)&ProdAmount, sizeof(double) );
os.write( (char*)&Tau, sizeof(double) );
os.write( (char*)&Order, sizeof(double) );
os.write( (char*)&logReact, sizeof(double) );
os.write( (char*)&logProd, sizeof(double) );
os.write( (char*)&lnRateConst, sizeof(double) );
os.write( (char*)&InverseTemp, sizeof(double) );
 }
 else     // delimated text
 {   // Delimator is the character to use as the delimator
long f = os.flags();
os.precision(13);
os.flags( f | ios::fixed);
 /*
char temp[2048];
ostrstream out(temp, 2048);
out << DatePerformed << Delimator << MassBalance << Delimator
<< TrapTime << Delimator
<< InjPress << Delimator << InjTempC << Delimator << InjTempK
<< Delimator << AmountInj << Delimator << ReactorTempC <<
Delimator
<< ReactorTempK << Delimator << ReactName << Delimator;
out << ReactChemForm << Delimator << ReactRetentTime <<
Delimator
<< ReactPeakArea << Delimator << ReactCoeffM << Delimator
<< ReactCoeffC << Delimator << ProdName << Delimator
<< ProdChemForm << Delimator << ProdRetentTime << Delimator
<< ProdPeakArea << Delimator << ProdCoeffM << Delimator
<< ProdCoeffC << Delimator;
out << ReactAmount << Delimator << ProdAmount << Delimator
<< logReact << Delimator << logProd << Delimator << Order
<< Delimator << AdjustedFlowRate  << Delimator << Tau
<< Delimator << RateConstant << Delimator << InverseTemp
<< Delimator << lnRateConst;
out.put('\0');
            MessageBox(0, temp, "Test", MB_OK );       */
 os << DatePerformed << Delimator << MassBalance << Delimator
<< TrapTime << Delimator
<< InjPress << Delimator << InjTempC << Delimator << InjTempK
<< Delimator << AmountInj << Delimator << ReactorTempC <<
Delimator
<< ReactorTempK << Delimator << ReactName << Delimator;
os << ReactChemForm << Delimator << ReactRetentTime <<
Delimator
<< ReactPeakArea << Delimator << ReactCoeffM << Delimator
<< ReactCoeffC << Delimator << ProdName << Delimator
<< ProdChemForm << Delimator << ProdRetentTime << Delimator
<< ProdPeakArea << Delimator << ProdCoeffM << Delimator
<< ProdCoeffC << Delimator;
os << ReactAmount << Delimator << ProdAmount << Delimator
<< logReact << Delimator << logProd << Delimator << Order
<< Delimator << AdjustedFlowRate  << Delimator << Tau
<< Delimator << RateConstant << Delimator << InverseTemp
<< Delimator << lnRateConst;
 }
return os;
 }

/*****************************************************************
KinetData member functions
*****************************************************************/
ofstream& KinetData::Save(ofstream& os, char Delimator)
//ostream& operator << (ostream& os, KinetData& d)
 {
unsigned char n = 0;
if( mode == KinetData::binary) // if internal flag is set to binary
 {
Data.Save(os, Delimator, ios::binary);
 }
 else     // delimated text
  {  // Delimator is the character to use as the delimator
 Data.Save(os, Delimator, ios::out);
 }
 Modified = FALSE;
 return os;
```

```
}
ifstream& KinetData::Load(ifstream& is, char Delimator)
//ifstream& operator >> (istream& is, KinetData& d)
{
unsigned char n = 0;
if( mode == KinetData::binary)   // file is in binary mode
 {
 is.read( (char*)&n, sizeof(unsigned char) );                    // read in
number of bytes in next string
 if( (n <= MAX) && (n) )
            {
is.read(Data.DatePerformed, n); // read in string if not too long
            }
else if(n) throw(FileReadErr( "DatePerformed", is.tellg(),
STRING_TOO_LONG));
// throw exception object with error data if string is too long
Data.DatePerformed[n] = '\0';
is.read( (char*)&Data.InjPress, sizeof(double) );
is.read( (char*)&Data.InjTempC, sizeof(double) );
is.read( (char*)&Data.InjTempK, sizeof(double) );
is.read( (char*)&Data.AmountInj, sizeof(double) );
is.read( (char*)&Data.TrapTime, sizeof(float) );
is.read( (char*)&Data.MassBalance, sizeof(double) );
is.read( (char*)&Data.ProdRetentTime, sizeof(double) );
is.read( (char*)&Data.ProdPeakArea, sizeof(double) );
is.read( (char*)&Data.ProdCoeffM, sizeof(double) );
is.read( (char*)&Data.ProdCoeffC, sizeof(double) );
is.read( (char*)&n, sizeof(unsigned char) );  // read in number of bytes
in next string
if( (n <= MAX) && (n) )
            {
is.read(Data.ProdName, n); // read in string if not too long
            }
else if(n) throw(FileReadErr( "ProdName", is.tellg(),
STRING_TOO_LONG, this)); // exit function if string is too long
Data.ProdName[n] = '\0';
is.read( (char*)&n, sizeof(unsigned char) ); // read in number of bytes
in next string
if( (n <= MAX) && (n) )
            {
is.read(Data.ProdChemForm, n); // read in string if not too long
            }
else if(n) throw(FileReadErr( "ProdChemForm", is.tellg(),
STRING_TOO_LONG, this)); // exit function if string is too long
Data.ProdChemForm[n] = '\0';
is.read( (char*)&Data.ReactRetentTime, sizeof(double) );
is.read( (char*)&Data.ReactPeakArea, sizeof(double) );
is.read( (char*)&Data.ReactCoeffM, sizeof(double) );
is.read( (char*)&Data.ReactCoeffC, sizeof(double) );
is.read( (char*)&n, sizeof(unsigned char) ); // read in number of bytes
in next string
if( (n <= MAX) && (n) )
            {
is.read(Data.ReactName, n); // read in string if not too long
            }
else if(n) throw(FileReadErr( "ReactName", is.tellg(),
STRING_TOO_LONG, this)); // exit function if string is too long
Data.ReactName[n] = '\0';
is.read( (char*)&n, sizeof(unsigned char) ); // read in number of bytes
in next string
if( (n <= MAX) && (n) )
            {
is.read(Data.ReactChemForm, n); // read in string if not too long
            }
else if(n) throw(FileReadErr( "ReactChemForm", is.tellg(),
STRING_TOO_LONG, this)); // exit function if string is too long
Data.ReactChemForm[n] = '\0';
is.read( (char*)&Data.AdjustedFlowRate, sizeof(double) );
is.read( (char*)&Data.ReactorTempC, sizeof(double) );
is.read( (char*)&Data.ReactorTempK, sizeof(double) );
is.read( (char*)&Data.RateConstant, sizeof(double) );
is.read( (char*)&Data.ReactAmount, sizeof(double) );
is.read( (char*)&Data.ProdAmount, sizeof(double) );
is.read( (char*)&Data.Tau, sizeof(double) );
is.read( (char*)&Data.Order, sizeof(double) );
is.read( (char*)&Data.logReact, sizeof(double) );
is.read( (char*)&Data.logProd, sizeof(double) );
is.read( (char*)&Data.lnRateConst, sizeof(double) );
is.read( (char*)&Data.InverseTemp, sizeof(double) );
 }
else  // file is not in binary mode
 {    // FileType is a member of TMainWindow. the Delimated
member of FileType is
// the character to use as the delimator
char temp[MAX+1];
int max = MAX+1;
is.ignore(3, Delimator);  // ignore first delimator in line
is.getline(Data.DatePerformed, max, Delimator);
is.getline(temp, max, Delimator);
Data.MassBalance = atof(temp);
is.getline(temp, max, Delimator);
Data.TrapTime = atof(temp);
is.getline(temp, max, Delimator);
Data.InjPress = atof(temp);
is.getline(temp, max, Delimator);
Data.InjTempC = atof(temp);
is.getline(temp, max, Delimator);
Data.InjTempK = atof(temp);
is.getline(temp, max, Delimator);
Data.AmountInj = atof(temp);
is.getline(temp, max, Delimator);
Data.ReactorTempC = atof(temp);
is.getline(temp, max, Delimator);
Data.ReactorTempK = atof(temp);
is.getline(Data.ReactName, max, Delimator);
is.getline(Data.ReactChemForm, max, Delimator);
is.getline(temp, max, Delimator);
Data.ReactRetentTime = atof(temp);
is.getline(temp, max, Delimator);
Data.ReactPeakArea = atof(temp);
is.getline(temp, max, Delimator);
Data.ReactCoeffM = atof(temp);
is.getline(temp, max, Delimator);
Data.ReactCoeffC = atof(temp);
is.getline(Data.ProdName, max, Delimator);
is.getline(Data.ProdChemForm, max, Delimator);
is.getline(temp, max, Delimator);
Data.ProdRetentTime = atof(temp);
is.getline(temp, max, Delimator);
Data.ProdPeakArea = atof(temp);
is.getline(temp, max, Delimator);
Data.ProdCoeffM = atof(temp);
is.getline(temp, max, Delimator);
Data.ProdCoeffC = atof(temp);
is.getline(temp, max, Delimator);
Data.ReactAmount = atof(temp);
is.getline(temp, max, Delimator);
Data.ProdAmount = atof(temp);
is.getline(temp, max, Delimator);
Data.logReact = atof(temp);
is.getline(temp, max, Delimator);
Data.logProd = atof(temp);
is.getline(temp, max, Delimator);
Data.Order = atof(temp);
is.getline(temp, max, Delimator);
Data.AdjustedFlowRate = atof(temp);
is.getline(temp, max, Delimator);
Data.Tau = atof(temp);
is.getline(temp, max, Delimator);
Data.RateConstant = atof(temp);
is.getline(temp, max, Delimator);
Data.InverseTemp = atof(temp);
is.getline(temp, max, Delimator);
Data.lnRateConst = atof(temp);
 }
Modified = TRUE;
return is;
}
```

```
void KinetData::Sprintf(char* str, int sz, FIELDS f )
{
switch (f)
 {
case (ALL) :
            {
ostrstream stream(str, sz);
stream << Data.DatePerformed << "\t" << Data.MassBalance << "\t"
<< Data.TrapTime
<< "\t" << Data.InjPress << "\t" << Data.InjTempC << "\t" <<
Data.InjTempK
<< "\t" << Data.AmountInj << "\t" << Data.ReactorTempC << "\t" <<
Data.ReactorTempK
<< "\t" << Data.ReactName << "\t" << Data.ReactChemForm << "\t"
<< Data.ReactRetentTime << "\t" << Data.ReactPeakArea << "\t"
<< Data.ReactCoeffM << "\t" << Data.ReactCoeffC << "\t" <<
Data.ProdName
<< "\t" << Data.ProdChemForm << "\t" << Data.ProdRetentTime <<
"\t"
<< Data.ProdPeakArea << "\t" << Data.ProdCoeffM << "\t" <<
Data.ProdCoeffC
<< "\t" << Data.ReactAmount << "\t" << Data.ProdAmount << "\t"
<< Data.logReact << "\t" << Data.logProd << "\t" << Data.Order << "\t"
<< Data.AdjustedFlowRate << "\t" << Data.Tau << "\t" <<
Data.RateConstant
<< "\t" << Data.InverseTemp << "\t" << Data.lnRateConst;
stream.put('\0');
break;
            }
case (REACT) : { break; }
case (PROD) : { break; }
default : { }
 }
}
void KinetData::Set(double order)
 {
Data.Order = order;
}
void KinetData::Set(TProdTransferBuffer& Prod)
 {
Modified = TRUE;
Data.Set(Prod);
CalcAmounts();    // automatically calculates after any changes
}
void KinetData::Set(TReactTransferBuffer&  React)
 {
Modified = TRUE;       // set whether modified flag to true
Data.Set(React);
CalcAmounts();    // automatically calculates after any changes
}
void KinetData::Reset()
 {
// flag (s)
Modified = FALSE; // whether or not anything has been changed since
the
// last saving
mode = binary;  // how to save or open a file
Data.Reset();
 }
KinetData::KinetData(TReactTransferBuffer&  React, save_flag m)
 {
 Reset();
Modified = TRUE;       // set whether modified flag to true
mode = m;
Set(React);
} // end of KinetData(TReactTransferBuffer& )
KinetData::KinetData(save_flag m)
 {
 Reset();
mode = m;
 }
 KinetData::KinetData(TProdTransferBuffer& Prod, save_flag m)
 {
 Reset();
mode = m;
Set(Prod);
}    // end of KinetData(TProdTransferBuffer& )
KinetData::KinetData(TReactTransferBuffer&  React,
TProdTransferBuffer& Prod, save_flag  m)
 {
Reset();
mode = m;
Set(Prod);
Set(React);
} // end of KinetData(TReactTransferBuffer& , TProdTransferBuffer& )
long KinetData::CheckEssData()
// check for presence of data essential to calculation
// for some pieces of data eg CoeffC  zero is accepted
 {
/* need to check for Reactor temp, pressinj (not really essential to calc
very important to the mass balance ratio), temp inj, reactant peak area,
product peak area, coeff's for prod and react
 */
long Missing = 0;
if( !Data.InjPress ) Missing |= INJPRESS;
 if(!Data.InjTempC)  Missing |= INJTEMP;
 if(!Data.ReactorTempC)  Missing |= REACTORTEMP;
if(!Data.ProdCoeffM)
Missing |= PRODCOEFFM;
 if(!Data.ReactCoeffM)  Missing |= REACTCOEFFM;
 if(!Data.ProdPeakArea)  Missing |= PRODPEAKAREA;
 if(!Data.ReactPeakArea)  Missing |= REACTPEAKAREA;
return Missing;
 }
long KinetData::CheckOtherData() // checks for presence of data not
essential to calculation
 {
long Missing = 0;
if( Data.DatePerformed[0] == '\0' ) Missing |= DATE;
if(!Data.TrapTime) Missing |= TRAPTIME;
if(!Data.ProdRetentTime) Missing |= PRODRETENT;
if(!Data.ReactRetentTime) Missing |= REACTRETENT;
if( Data.ProdName[0] == '\0' ) Missing |= PRODNAME;
if( Data.ProdChemForm[0] == '\0' ) Missing |= PRODFORM;
if( Data.ReactName[0] == '\0' ) Missing |= REACTNAME;
if( Data.ReactChemForm[0] == '\0' ) Missing |= REACTFORM;
return Missing;
 }
int KinetData::CalcAmounts()
 {
if(CheckEssData() ) return 0;  // not enough data to calculate
// temperature conversions
Data.InjTempK = Data.InjTempC + CinK;
Data.ReactorTempK = Data.ReactorTempC + CinK;
// temp calculations
Data.InverseTemp = 1 / Data.ReactorTempK;
// calc amount injected from n = PV / RT
Data.AmountInj = ( (Data.InjPress * TORRtoPA) * SAMPLELOOP)
                                                    /
(R * Data.InjTempK);
// calc amounts detected from CoeffC and CoeffM
Data.ReactAmount = (Data.ReactPeakArea - Data.ReactCoeffC) /
Data.ReactCoeffM;
Data.ProdAmount = (Data.ProdPeakArea - Data.ProdCoeffC) /
Data.ProdCoeffM;
if(Data.ReactAmount > 0)  Data.logReact = log10(Data.ReactAmount);
if(Data.ProdAmount > 0)  Data.logProd = log10(Data.ProdAmount);
// calc mass balance from amounts detected and injected
Data.MassBalance = (Data.ReactAmount + Data.ProdAmount) /
Data.AmountInj;
long flag = Config.CheckData();
if( (flag & (~DETECTOR)) )  // if anything besides the detector
information is missing
  {
return 2;  // full calculation not done as not all information there
  }
 else
  {
```

```
// U = Ua Tr Pa / Ta Pr
Data.AdjustedFlowRate = (Config.FlowRate * ATMOSPRESSpsi *
Data.ReactorTempK)
/       // reactor press must made absolute
// it is a relative press as entered
// hence ATMOSPRESSpsi is added
( (Config.MeasTemp + CinK) * (Config.ReactorPress +
ATMOSPRESSpsi));
Data.Tau = Config.ReactorVol / Data.AdjustedFlowRate;
double temp;
if( (temp = pow(Config.ReactorVol, (Data.Order-1))) != HUGE_VAL)
// check result not out of range
            {
if( (errno != ERANGE) && (errno != EDOM) )
// check range flag again and check for domain error
            {
double temp2;   // do checks again for this power calc
if( (temp2 = pow( Data.ReactAmount, Data.Order)) != HUGE_VAL)
            {
if( (errno != ERANGE) && (errno != EDOM) )
                        {
if(temp2 * Data.Tau)  // to stop divide by zero errors
                        {
Data.RateConstant = ( Data.ProdAmount * temp * Data.Order )
                                    /
(temp2 * Data.Tau);
if(Data.RateConstant > 0) Data.lnRateConst =
log(Data.RateConstant);
} // end of if
} // end of if
                }  // end of if
                }  // end of if
                }  // end of if
} // end of else from Config.CheckData()
return 1; // successfully calculated
}    // end of CalcAmounts()
```

Kinet1.cpp

```
#include "kinet1a.h"
#include "kinet1b.h"
#include "genfunct.h"
 // global variables initialisation
TConfigData KinetData::Config;     // static member of KinetData
char Delimator;
// returns 0 if there is a problem opening the file and displays
// a mesage box telling the user this
int TMainWindow::GoodOpen( const char* filename, ofstream& os,
ios::open_mode mode)
            {
os.open(filename, mode);
if(!os)
            {
throw( FileReadErr(filename, 0, UNABLE_TO_OPEN ));
            }
else return 1;
            }
int TMainWindow::GoodOpen( const char* filename, ifstream& is,
ios::open_mode mode)
            {
is.open(filename, mode | ios::nocreate);
if(!is)
            {
MessageBox("Unable to open file", "File Error", MB_OK |
MB_ICONEXCLAMATION);
return 0;
            }
else return 1;
            }
DEFINE_RESPONSE_TABLE1(TMainWindow, TWindow)
EV_WM_RBUTTONDOWN,
EV_WM_LBUTTONDOWN,
EV_WM_LBUTTONDBLCLK,
EV_WM_HSCROLL,
EV_WM_VSCROLL,
EV_WM_MOVE,
EV_WM_SIZE,
EV_COMMAND(CM_DEBUG_INFO, DebugInfo),
EV_COMMAND(CM_CONFIG, HandleCMConfig),
EV_COMMAND(CM_FILE_CLOSE, HandleClose),
EV_COMMAND(CM_FILE_OPEN, HandleCMFileOpen),
EV_COMMAND(CM_FILE_SAVEAS, HandleCMFileSaveAs),
EV_COMMAND(CM_FILE_SAVE, HandleCMFileSave),
// EV_COMMAND(CM_IMPORT, HandleImport),
EV_COMMAND(CM_EXPORT, HandleExport),
EV_COMMAND(CM_EXIT, HandleCMExit),
EV_COMMAND(CM_EDIT_ORDER, EditOrder),
EV_COMMAND(CM_DATA_NEWKIN, HandleCMDNewKin),
EV_COMMAND(CM_DATA_NEWREACT, HandleNewReact),
EV_COMMAND(CM_DATA_NEWPROD, HandleNewProd),
EV_COMMAND(CM_DATA_EDIT, HandleCMDEdit),
EV_COMMAND(CM_DATA_DELETE, HandleDelete),
EV_COMMAND(CM_CALC_AMOUNTS, CalcAmounts),
EV_COMMAND(CM_CALC_ORDER, CalcOrder),
EV_COMMAND(CM_CALC_CALPROD, HandleCalCalcProd),
EV_COMMAND(CM_CALC_CALREACT, HandleCalCalcReact),
EV_COMMAND(CM_CALC_ACTENERGY, CalcActEngy),
EV_COMMAND(CM_VIEW_PRODUCT, HandleViewProd),
EV_COMMAND(CM_VIEW_REACTANT, HandleViewReact),
EV_COMMAND(CM_VIEWKINETIC_DATA,
HandleViewKinetic_Data),
END_RESPONSE_TABLE;
/*------------------------------------------------------------
INITIALISATION        AND        SETUP
------------------------------------------------------------*/
TMainWindow::TMainWindow()
: TWindow(0, 0, 0)
            {
KineticTitleExtent.cy = 0;
KineticTitleExtent.cx = 0;
```

```
ProdReactTitleExtent.cy = 0;
ProdReactTitleExtent.cx = 0;
KineticLocOrigin.x = 0;
KineticLocOrigin.y = 0;
ReactLocOrigin.x = 0;
ReactLocOrigin.y = 0;
ProdLocOrigin.x = 0;
ProdLocOrigin.y = 0;
Colour = new TColor(TColor::LtBlue);
Font = new TFont("Arial", 15, 9);
KinData = new TKinData(10, 0, 10);
KinSelected = new TLongArray(10, 0 ,10);
ReactSelected = new TLongArray(10, 0 ,10);
ProdSelected  = new TLongArray(10, 0 ,10);
FileData = new TOpenSaveDialog::TData(
DWORD(OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT), // flags
OpenSaveFilter,    // filter string declared at beginning of file
                                                    "*.kin",
// custom filter
            0,          // initial directory
            "*.kin");     // default extension
// set config data to defaults
ConfigData.ReactorVol = DefReactorVol;
ConfigData.GasConstant = DefGasConstant;
strcpy(ConfigData.Detector, DefDetector);
// set modified to FALSE
Modified(FALSE);
IsNewFile = TRUE;
        }
void TMainWindow::SetupWindow()
{
TWindow::SetupWindow();  // create child controls
Attr.Style |= WS_HSCROLL | WS_VSCROLL |
WS_THICKFRAME;
UpdateScrollData(TRUE);
strcpy(FileType.version, Version); // initialise version data in filetype
CalView = KINETIC;  // start off in calibration reactant view
// setup static member of KinetData inline with information in program
KinetData::Config = ConfigData;
}
/*_____________________________________________
FILE COMMANDS
_____________________________________________*/
/*
void TMainWindow::HandleImport()
{
if( CanClose() )
 {
Close();
FileData->SetFilter(ImportExportFilter);
strcpy(FileData->DefExt, ExpExt); // set to default extension for import
files
if( !("(FileData->FileName) == '\0') )    // if there is a filename
            {
char name[MAXFILE];
fnsplit(FileData->FileName, 0, 0, name, 0);
strcpy(FileData->FileName, name);
strcat(FileData->FileName, ExpExt); // build filename with default
import extension
            }
else
            {
strcpy(FileData->FileName, "*.");    // there is no filename
strcat(FileData->FileName, ExpExt);
            }
if(TFileOpenDialog(this, *FileData).Execute() == IDOK)
            {
if(InputTrans.InputData[0] == 0)  // if transfer buffer is empty
                        {
InputTrans.InputData[0] = ","; // put quote character into transfer buffer
                        }
Delimator = FileType.Delimator = '\t'; // set delimator then check alright
if(MessageBox( "OK to use tab as delimator?", "Import File",
MB_YESNO | MB_ICONQUESTION) == IDNO ) // want to enter a
new delimator
            {
if(TInputBox(this, DLG_INPUT).Execute() == IDOK)
                        {
if(InputTrans.InputData[0] == 0)  // if a blank was input
                        {
// put quote character into transfer buffer
InputTrans.InputData[0] = ",";
                        }
Delimator = FileType.Delimator = InputTrans.InputData[0]; // set
delimator
Import(); // import data with new delimator
                        }
            }
Import();  // no change to the delimator
            }
 }
} // end of HandleImport()
void TMainWindow::Import()
{
try   // try whole function
 {
KinetData* tmp;
ifstream is;
FileType.Reset();                  // reset filetype data
FileType.Binary = KinetData::delimated;  // set to correct file type
if(!GoodOpen(FileData->FileName, is, ios::in) )  // in text mode
            {
Close();  // reset window for new document
return; // exit function
            }
if( (KinData->GetItemsInContainer() ) > 0)
KinData->Flush( TShouldDelete::Delete);
// if at this point no exception has been thrown so file is correct
 is.seekg(0, ios::beg);      // set file pointer to beginning of file
 is >> ConfigData;  // read config data
is.ignore(5, '\n');
is.ignore(600, '\n');  // bypass title line before loading data
while(is.peek() !=EOF)
            {
tmp = new KinetData(FileType.Binary);
tmp->Load(is, FileType.Delimator); // read in from file
KinData->Add(tmp);
is.ignore(1, '\n');
            }

FileType.NumRecsKin = KinData->GetItemsInContainer();   // set to
correct number of records
Modified(TRUE);
IsNewFile = TRUE;
if(KineticTitleExtent.cy == 0)
            {                  // if no title has been displayed yet( therefore
title height
            KineticLocOrigin.x = 0;      // is unknown) set origin to 0,0
KineticLocOrigin.y = 0;
            }
else
{      // if Title height is known set origin to just below the title
KineticLocOrigin.x = 0;
KineticLocOrigin.y = KineticTitleExtent.cy;
            }
CalViewMode(KINETIC); // set view mode to kinetic
Invalidate();
UpdateWindow();
UpdateScrollData();
} // end of try block
catch(FileReadErr e)
 {
char message[100];
ostrstream stream(message, 100);
if(e.fault == UNABLE_TO_OPEN)
            {
stream << FileData->FileName << " is not the correct format"
```

```
<< endl << e.str;
stream.put('\0');
MessageBox(message, "File Error", MB_OK |
MB_ICONEXCLAMATION);
            }
else
            {
stream << "File Open Interrupted!  Error in " << e.str << " field read at
file position"
<< e.FilePointer << "." << endl;
if(e.fault == STRING_TOO_LONG) stream << "String too long" ;
else if(e.fault == UNKNOWN) stream << "Unknown error" ;
else if(e.fault == DISK_ERROR) stream << "Disk Error";
if( (e.fault != UNABLE_TO_OPEN) && (e.fault !=DISK_ERROR) )
stream << "\nThis is not in a known Kin Calc file format";
MessageBox(message, "File Error", MB_OK |
MB_ICONEXCLAMATION);
if(e.tmp) delete e.tmp;  // free memory allocated to incomplete object
IsNewFile = TRUE;  // so that file has not in effect loaded
Modified(FALSE);
  }
}// end of catch
}
*/
void TMainWindow::HandleExport()
{
if(!KinData->IsEmpty())
 {
if( CanClose() )
            {
FileData->SetFilter(ImportExportFilter);
strcpy(FileData->DefExt, ExpExt); // set to default extension for export
files
if( !(*(FileData->FileName) == '\0') )    // if there is a filename
            {
char name[MAXFILE];
fnsplit(FileData->FileName, 0, 0, name, 0);
strcpy(FileData->FileName, name);
strcat(FileData->FileName, ExpExt);  // build filename with default
export extension
            }
else
            {
strcpy(FileData->FileName, "*" );  // there is no filename
strcat(FileData->FileName, ExpExt); // build mask with default export
extension
            }
if(TFileSaveDialog(this, *FileData).Execute() == IDOK)
            {
if(InputTrans.InputData[0] == 0)  // if transfer buffer is empty
                        {
InputTrans.InputData[0] = "";  // put quote character into transfer buffer
                        }
Delimator = FileType.Delimator = '\t'; // set delimator then check alright
if(MessageBox( "OK to use tab as delimator?", "Export File",
MB_YESNO | MB_ICONQUESTION) == IDNO ) // want to enter a
new delimator
                        {
if(TInputBox(this, DLG_INPUT).Execute() == IDOK)
                        {
if(InputTrans.InputData[0] == 0)  // if a blank was input
                        {
// put quote character into transfer buffer
InputTrans.InputData[0] = "";
                        }
Delimator = FileType.Delimator = InputTrans.InputData[0]; // set
delimator
Export(); // export data
                        }
                        }
Export();  // no change in delimator
            }
} // end of if(CanClose())
} // end of if(KinData->IsEmpty)
}
void TMainWindow::Export()   // puts one delimator before data
{
ofstream os;
TKinDataIterator i(*(KinData) );
char* title;
FileType.Binary = KinetData::delimated;
if(!GoodOpen(FileData->FileName, os, ios::out) )  // open in text mode
 {
return; // exit function
 }
/*********************
write configData then newline
 *********************/
// to copy title to so tabs can be replaced with Delimator
title = new char[strlen(ConfigTitleText) + 1];
strcpy(title, ConfigTitleText);
CharReplace(title, '\t', FileType.Delimator);  // put delimator into title
instead of tabs
os << FileType.Delimator << "ConfigData\n"
<< FileType.Delimator << title << "\n"
<< FileType.Delimator;
ConfigData.Save(os, Delimator, ios::out );
os << "\n";
delete title;     // deallocate memory used for title editing
/*************************************
Write Reactant cal data
 *********************/
os << FileType.Delimator << "Reactant Calibration Data\n" <<
Delimator;
ReactCalData.Save(os,  FileType.Delimator, ios::out);
/*************************************
Write product cal data
 *********************/
os << FileType.Delimator << "Product Calibration Data\n" <<
Delimator;
ProdCalData.Save(os,  FileType.Delimator, ios::out);
/*************************************
Write Kinetic data
 *********************/
os << "\n\n\n" << FileType.Delimator  << "KINETIC DATA\n\n";
title = new char[strlen(TArrheniusTitle) + 1];
strcpy(title, TArrheniusTitle);
CharReplace(title, '\t', FileType.Delimator);
os << FileType.Delimator << title << "\n" << Delimator;
KinArr.Save(os, ios::out, FileType.Delimator);
delete title;
os << "\n\n";
// write  titles
int f = strlen(KineticTitleText);  // text used to put titles onscreen
title = new char[f+1];
strcpy(title, KineticTitleText);
// must replace tabs in TitleText with Delimator
CharReplace(title, '\t', Delimator);
os << FileType.Delimator << title << "\n";  // write title to first line
// write data records
while(i)
 {
os << FileType.Delimator;
(i.Current())->Mode(FileType.Binary);
(i++)->Save(os, FileType.Delimator);
os << "\n";    // iterate through data
 }
delete title; // deallocate memory
// do not bother changing modified or isnewfile status as it is desirable
that
// saving in this format is not considered as saving proper
}
void TMainWindow::HandleCMExit()
{
// send a WM_CLOSE message to the parent window
Parent->SendMessage(WM_CLOSE);
}
int TMainWindow::WriteFile()  // always in binary
```

```
{
ofstream os;
TKinDataIterator i(*(KinData) );
FileType.NumRecsKin =  KinData->GetItemsInContainer();
FileType.NumRecsReact =
ReactCalData.data().GetItemsInContainer();
FileType.NumRecsProd = ProdCalData.data().GetItemsInContainer();
strcpy(FileType.version, Version);
FileType.Binary = KinetData::binary;
if(!GoodOpen(FileData->FileName, os, ios::binary) ) // open in binary
mode
 {
return 0; // exit function
 }
 os.write( (char*)&FileType, sizeof(TFileType) );  // save file type data
first
 os.write( (char*)&ConfigData, sizeof(ConfigData) ); // save config data
 // write cal data
 ReactCalData.Save(os, 0, ios::binary);  // write react cal data in binary
mode; no delimator needed
 ProdCalData.Save(os, 0, ios::binary);  // write prod cal data in binary
mode
// write kinetic records
while(i)
 {
(i.Current())->Mode(FileType.Binary);
(i++)->Save(os, 0);    // iterate through data
 }
// save arrhenius data
KinArr.Save(os, ios::binary, 0);
Modified(FALSE);
IsNewFile = FALSE;
return 1;
}
void TMainWindow::HandleCMFileSaveAs()
 {
if( (!KinData->IsEmpty()) || (!ReactCalData.IsEmpty() ) // if there is data
to save
|| ( !ProdCalData.IsEmpty() ) )
 {
char name[MAXFILE];
FileData->SetFilter(OpenSaveFilter);
if(IsNewFile)
            {
strcpy(FileData->FileName, "*");    // there is no filename
strcat(FileData->FileName, OpenSaveExt);
            }
else
            {
fnsplit(FileData->FileName, 0, 0, name, 0);
strcpy(FileData->FileName, name);
strcat(FileData->FileName, OpenSaveExt);
            }
if(TFileSaveDialog(this, *FileData).Execute() == IDOK)
            {
if(WriteFile() )
            {
int s; // for string length
char text[MaxEditLen];  // character string for building title
ostrstream out(text, MaxEditLen-1); // string stream for outputting to text
char drive[MAXDRIVE]; // drive
char dir[MAXDIR];  // path
char name[MAXFILE]; // filename
char ext[MAXEXT];  // extension (includes leading .)
int l = 0;  // for string search, counting of characters
char *ptr;
int flag;
if((s = strlen(FileData->FileName)) > 30) // long filename
                        {
flag = fnsplit(FileData->FileName, drive, dir, name, ext);
ptr = dir; // set pointing to directory information
while(ptr)  // count number of \ in directory name
                        {
ptr = strchr(ptr + 1, '\\');  // search for next \ in path
l++;
                        }
                        if( l < 4)   // two or less directory names
strcpy(text, FileData->FileName);   // use whole filename
                        else if( l > 3 )    // 3 or more directory names
                        {  // of form  \name1\name2\name3\  only
want last 2 names
            ptr = dir;
            for(int i = 0; i != l-3; i++)  // go on until 2 from end
                        {
ptr = strchr(ptr + 1, '\\');
                        }
// put drive, some dots then the rest of the name into the output string
out << drive << "\\..." << ptr << name << ext;
out.put('\0');
} // end of else if
}  // end of if strlen >30
else  // use whole filename
                        {
out << FileData->FileName;
out.put('\0');
                        }
SetDocTitle(text,0);
              }
              }
 }
}
void TMainWindow::HandleCMFileSave()
{
if( (!KinData->IsEmpty()) || (!ReactCalData.IsEmpty() ) // if there is data
to save
|| ( !ProdCalData.IsEmpty() ) )
 {
if(IsNewFile) HandleCMFileSaveAs();
else WriteFile();
 }
}
void TMainWindow::HandleCMFileOpen()
 {
if( CanClose() )
 {
FileData->SetFilter(OpenSaveFilter);
strcpy(FileData->DefExt, OpenSaveExt); // set to default extension for
data files
strcpy(FileData->FileName, "*");    // there is no filename
strcat(FileData->FileName, OpenSaveExt);
if(TFileOpenDialog(this, *FileData).Execute() == IDOK)
if(OpenFile()) // file opened OK
            {
int s; // for string length
char text[MaxEditLen];  // character string for building title
ostrstream out(text, MaxEditLen-1); // string stream for outputting to text
char drive[MAXDRIVE]; // drive
char dir[MAXDIR];  // path
char name[MAXFILE]; // filename
char ext[MAXEXT];  // extension (includes leading .)
int l = 0;  // for string search, counting of characters
char *ptr;
int flag;
if((s = strlen(FileData->FileName)) > 30) // long filename
                        {
flag = fnsplit(FileData->FileName, drive, dir, name, ext);
ptr = dir; // set pointing to directory information
while(ptr)  // count number of \ in directory name
                        {
ptr = strchr(ptr + 1, '\\'); // search for next \ in path
l++;
                        }
                        if( l < 4)   // two or less directory names
            strcpy(text, FileData->FileName);   // use whole filename
                        else if( l > 3 )    // 3 or more directory names
                        {  // of form  \name1\name2\name3\  only
want last 2 names
                        ptr = dir;
```

```
                        for(int i = 0; i != i-3; i++) // go on until 2 from
end
                        {
ptr = strchr(ptr + 1, '\\');
                        }
// put drive, some dots then the rest of the name into the output string
out << drive << "\\..." << ptr << name << ext;
out.put('\0');
} // end of else if
} // end of if strlen >30
else // use whole filename
                        {
out << FileData->FileName;
out.put('\0');
                        }
SetDocTitle(text,0);
} // end of if ( Open() )
 }
}
int TMainWindow::OpenFile()  // always open in binary
{
try
{   // try whole function
Close();
ifstream is;
if(!GoodOpen(FileData->FileName, is, ios::binary) ) // binary mode
            {
Close(); // reset window for new document
return 0; // exit function
            }
KinetData* tmp;
streampos size = 0;
is.read( (char*)&FileType, sizeof(FileType));// read in FileType data
if( (FileType.Binary != KinetData::binary ) ||  // wrong format
(strcmp(FileType.version, Version)) ) // wrong version  or format
// this is an error condition.
            {
throw(string("Incorrect format") );
            }
// read config data
is.read((char*)&ConfigData, sizeof(ConfigData));
KinetData::Config = ConfigData;  // update kinetdata's copy
// read calibration data
ReactCalData.Load(is, 0,  FileType.NumRecsReact, ios::binary);
ProdCalData.Load(is, 0,  FileType.NumRecsProd, ios::binary);
for(int i=0; i<FileType.NumRecsKin; i++) // for all records
                        {
tmp = new KinetData(FileType.Binary);
                        tmp->Load(is, FileType.Delimator);     //
delimator is not
                        // used here but is a required argument for this
function
                        KinData->Add(tmp);
                        }
// load arrhenius data
KinArr.Load(is, ios::binary, 0);
Modified(FALSE);
IsNewFile = FALSE;
if(KineticTitleExtent.cy == 0)
            {                   // if no title has been displayed yet( therefore
title height
            KineticLocOrigin.x = 0;      // is unknown) set origin to 0,0
KineticLocOrigin.y = 0;
            }
else
{      // if Title height is known set origin to just below the title
KineticLocOrigin.x = 0;
KineticLocOrigin.y = KineticTitleExtent.cy;
            }
// initial value of longest lines as they will not have been drawn yet
LongestLineReact = LongestLineProd =
LongestLineKin = (GetClientRect().Width()) * 1.1;
CalViewMode(KINETIC); // set view mode to kinetic
return 1; // opened alright
} // end of try block
catch(FileReadErr& e)    // catch file exceptions generated when
loading data
            {
char message[100];
ostrstream stream(message, 100);
stream << "File Open Interrupted!  Error in " << e.str << " field read at
file position"
<< e.FilePointer << "." << endl;
if(e.fault == STRING_TOO_LONG) stream << "String too long" ;
else if(e.fault == UNKNOWN) stream << "Unknown error" ;
else if(e.fault == DISK_ERROR) stream << "Disk Error";
else if(e.fault == UNABLE_TO_OPEN) stream << "Unable to Open";
if( (e.fault != UNABLE_TO_OPEN) && (e.fault !=DISK_ERROR) )
stream << "\nThis is not in a known Kin Calc file format";
stream.put('\0'); // end string in stream
MessageBox(message, "File Error", MB_OK |
MB_ICONEXCLAMATION);
Close();  // so that file has not in effect loaded
return 0;  // not opened alright
} // end of catch
catch( string& s )
 {
char mess[MaxEditLen];
ostrstream str( mess, MaxEditLen );
if(s.length() > MaxEditLen)
MessageBox("Unable to open file", "File Error", MB_OK |
MB_ICONEXCLAMATION);
else
            {
str << "Unable to open file" << "\n" << s;
str.put('\0');
MessageBox(mess, "File Error", MB_OK |
MB_ICONEXCLAMATION);
            }
} // end of catch(string)
}
void TMainWindow::HandleClose()
{
if( CanClose() )   Close();
}
void TMainWindow::Close()
{
if( !KinData->IsEmpty())
KinData->Flush( TShouldDelete::Delete);    // clear kinetic data
if( !KinSelected->IsEmpty() )
KinSelected->Flush();  // clear kinetic selected list
if( !ReactSelected->IsEmpty() ) ReactSelected->Flush(); // clear
reactant selected list
if( !ProdSelected->IsEmpty() )  ProdSelected->Flush(); // clear product
selected list
 ProdCalData.Clear();   // clear product calibration data
 ReactCalData.Clear();   // clear reactant calibration data
KinRegress.reset();  // reset regression data from kinetic view
TWindow::SetDocTitle("Untitled", 0);
Modified(FALSE);
IsNewFile = TRUE;
FileType.Reset();
ResetDataOrg();
Invalidate();
UpdateWindow();
UpdateScrollData();
}
/*_____________________________________________
DISLPAY AND INTERNAL FUNCTIONS
_____________________________________________*/
void TMainWindow::Paint(TDC& tdc, BOOL erase, TRect&)
{
switch (CalView)
 {
case KINETIC: {        // do kinetic data
PaintKinetic( tdc,  erase);
break;
                                                    }
```

```
default : {
PaintReact(tdc, erase, CalView);
            }
 }
}

void TMainWindow::PaintKinetic(TDC& tdc, BOOL erase)
{
long lines = KinData->GetItemsInContainer();
LongestLineKin = 0; // reset for recalculation
TRect rect = GetClientRect();
height = rect.Height();
width = rect.Width();
static BOOL FirstTitle = TRUE;
// has the title been drawn yet since the program started
// for tabs for coefficient and Arrhenius text
int coTab[NumRegrTitTab + NumTArrheniusTitTab ];
BOOL coeff_flag = TRUE;
TEXTMETRIC metric;
tdc.SelectObject(*Font);
tdc.GetTextMetrics(metric);
KineticHScroll.LineMagnitude = metric.tmAveCharWidth +
metric.tmOverhang;
LineHeight = (metric.tmHeight + metric.tmExternalLeading);
if( lines ) // if there is data to paint
 {
char str[BufLen]; // to build output string in
unsigned int start, finish;  // index for range of data to output
TKinDataIterator i(*KinData);
 TRect r;       // for drawing selected rectangles
 UINT rectStart, rectFinish;  // indexes of which rectangles to draw
TPoint Offset;
TSize TextExtent;
// calculate starting position
TPoint pos(KineticLocOrigin);
// modify rect to allow room to paint coefficients
rect.bottom -= 3 * LineHeight;
if(rect.Height() < 3 * LineHeight)
            {
coeff_flag = FALSE;
rect.bottom += 3 * LineHeight;
rect.Normalize();
            }
// calculate tab length
int tab[KineticNumTabs];
for(int j=0; j < KineticNumTabs; j++)
            {
if(j==0)
            {
tab[j] = (metric.tmAveCharWidth * KinetictabChar[j]);
continue;
            }
tab[j] = (metric.tmAveCharWidth * KinetictabChar[j]) + tab[j-1]; //
calculate each tab in array
            }
// draw title then modify rect to exclude this area
tdc.TabbedTextOut(
            TPoint(pos.x, 0),       // starting coordinates
            KineticTitleText,          // address of text
            strlen(KineticTitleText),      // number of characters
            sizeof(tab)/ sizeof(int),  // number of tabs in array  (if 1 all
tabs tab[0] apart)
            tab,                // array for tab positions
            pos.x,              // x-coord for tab expansion
KineticTitleExtent);
// information about size of title on the screen is put in here
LongestLineKin = KineticTitleExtent.cx;
 if(FirstTitle)  // if the title was just drawn for the first time
            {          // need to adjust local origin to account for this
KineticLocOrigin.y = pos.y = KineticTitleExtent.cy;
            }
// set pen colour to background colour of TDC object if need to
// manually erase clear parts of window
if(erase == FALSE)
            {
tdc.SelectObject( TPen(tdc.GetBkColor()) );
r.Set(pos.x + KineticTitleExtent.cx, rect.top, rect.right, rect.top +
KineticTitleExtent.cy);
tdc.Rectangle(r);
            }
// modify rectangle for presence of title
rect.top += KineticTitleExtent.cy;
if(pos.y < rect.top) pos.y = rect.top;
if( (rect.right < KineticLocOrigin.x)
|| (rect.bottom < KineticLocOrigin.y) ) return;
// exit function if text is offscreen
Offset = KineticLocOrigin - TPoint(rect.left, rect.top);  // calculate offset
if( Offset.y < 0 ) // if KineticLocOrigin is above top of rect
            {
Offset.y = abs(Offset.y);
start = (Offset.y / LineHeight);
finish = (( rect.bottom - rect.top ) / LineHeight ) + start+1;  // length of
screen past start
            }
else
            {
start = 0;
finish = ( (rect.bottom - KineticLocOrigin.y) / LineHeight)+1;
            }
if(finish > lines) finish = lines;   // stay within range of data array
i.Restart(start, finish); // set iterators range
// draw text
while(i)  // for whole range to draw
            {
i++->Sprintf(str,BufLen);  // build string
tdc.TabbedTextOut(
            pos,          // starting coordinates
            str,          // address of text
            strlen(str),     // number of characters
            sizeof(tab)/ sizeof(int),  // number of tabs in array   (if 1 all
tabs tab[0] apart)
            tab,          // array for tab positions
            pos.x,          // x-coord. for tab expanding
            TextExtent);    // information about size of text on the
screen is put in here
           if(erase == FALSE)  // if rect not erased first clear off area
not written in
            {
if( (pos.x + TextExtent.cx) < width)
            {
            r.Set(pos.x + TextExtent.cx,    // end of string
                                    pos.y,          // row of text
                                    width,          // right hand edge of
client area
(pos.OffsetBy(0,LineHeight)).y); // down 1 line
tdc.Rectangle(r); // draw blank rectangle
            }
            }
pos.Offset(0, LineHeight);   // move down a line
if(TextExtent.cx > LongestLineKin) LongestLineKin = TextExtent.cx;
            }
if(erase==FALSE)
            {
if(pos.y < height)
            {
r.Set( 0,
pos.y,
width,
height);      // set rectangle to clear rest of screen if empty
tdc.Rectangle(r); // draw rectangle r in background colour, already set in
loop above
            }
            }
// if flag is set to draw coeff data, i.e. is there room?
if(coeff_flag)
{ // draw coefficient text
coTab[0] = LinRegrTitTab[0] * metric.tmAveCharWidth;          // calc
tabs
```

```
for(int o=1; o < NumRegrTitTab; o++)
                {
coTab[o] = (LinRegrTitTab[o] * metric.tmAveCharWidth) + coTab[o-
1];
                }
for( o=NumRegrTitTab; o < NumRegrTitTab +
NumTArrheniusTitTab; o++)
                {
coTab[o] = (TArrheniusTitTab[o - NumRegrTitTab ]
metric.tmAveCharWidth) + coTab[o-1];
                }
pos.y = rect.bottom + LineHeight;
int TitSz = strlen(LinRegressTitle) + strlen(TArrheniusTitle) + 20;
char *coefftit = new char[TitSz];
ostrstream out(coefftit, TitSz); // build title for one off data
out << "\t" << LinRegressTitle << "\t" << TArrheniusTitle
                        << "\tKinetic Data";     // append view status
        out.put('\0');              // end string
tdc.TabbedTextOut(
                pos,          // starting coordinates
                coefftit,          // address of text
                strlen(coefftit),     // number of characters
                sizeof(coTab)/ sizeof(int), // number of tabs in array   (if 1
all tabs tab[0] apart)
                coTab,          // array for tab positions
                pos.x,          // x-coord. for tab expanding
                TextExtent);     // information about size of text on the
screen is put in here
delete coefftit;
KinRegress.Sprintf(str, BufLen-1); // build string for output
strcat(str, "\t"); // add a tab onto the end
// output KinArr into str allowing for KinRegress already being in it
KinArr.Sprintf(str + strlen(str), BufLen - strlen(str) );
pos.Offset(0, LineHeight);   // move printing position
tdc.TabbedTextOut(
                pos,          // starting coordinates
                str,          // address of text
                strlen(str),     // number of characters
sizeof(coTab)/ sizeof(int),
// number of tabs in array   (if 1 all tabs tab[0] apart)
                coTab,          // array for tab positions
                pos.x,          // x-coord. for tab expanding
                TextExtent);     // information about size of text on the
screen is put in here
  }
  /* ************************************************************
Rectangle drawing part
  ************************************************************/
if(!KinSelected->IsEmpty())  // some records are selected
                {
long numSel = KinSelected->GetItemsInContainer();
for(long m = 0; m <= numSel -1; m++) // decide which selected
records are onscreen
                {
if( ((*KinSelected)[m] <= start) &&  ((*KinSelected)[m+1] > start) )
rectStart = m;
if(finish >= numSel) rectFinish =numSel;
else if( ((*KinSelected)[m] <= finish) && ((*KinSelected)[m+1] > finish))
rectFinish = m+1;
                }
// commented out routine to decide on visible records for debugging of
the rest of the
// function.  Temporary few lines that set indexes to draw all selected
records
rectStart = 0;
rectFinish = numSel-1;
for(m = rectStart; m <= rectFinish; m++)
                {
r.Set(0,
KineticLocOrigin.y + ((*KinSelected)[m]) * LineHeight,
rect.Width(),
KineticLocOrigin.y + ((*KinSelected)[m] * LineHeight) +
(LineHeight*1.05) );
if(!tdc.InvertRect(r)) return;  // exit Paint if rectangle draw fails
                }
}       // end of if(!KinSelected->IsEmpty())
 /* ************************************************************
End of Rectangle drawing part
 ************************************************************/
FirstTitle = FALSE; // the title has been drawn at least once
} // end of if(lines)
} // end of PaintKinetic
void TMainWindow::PaintReact(TDC& tdc, BOOL erase, long View)
{
TProductReactantCal *d;  // for data
TLongArray *Selected;        // for selected list
TPoint *p;           // for origin
int *LL;             // for longest line
TScrollData *H;     // for scroll data
static BOOL ProdFirstTitle = TRUE;  //  has the title been drawn yet
since the program started
static BOOL ReactFirstTitle = TRUE;
BOOL* FirstT;
switch (View)
 {
case REACTANT :
                        {
d = &ReactCalData;
Selected = ReactSelected;
p = &ReactLocOrigin;
LL = &LongestLineReact;
H = &ReactHScroll;
FirstT = &ReactFirstTitle;
break;
                        }
case PRODUCT :
                        {
d = &ProdCalData;
Selected = ProdSelected;
p = &ProdLocOrigin;
LL = &LongestLineProd;
H = &ProdHScroll;
FirstT = &ProdFirstTitle;
break;
                        }
 }
// debugging part
long sel[400];
for(int h=0; h< Selected->GetItemsInContainer(); h++)
 {
sel[h] = (*Selected)[h];
 }
// end of debugging insert
TProductReactantCal& Data = *d;  // for data
 TPoint& Origin = *p;           // for origin
 int& LongestLine = *LL;            // for longest line
TScrollData& HScroll = *H;
BOOL FirstTitle = *FirstT;
long lines = Data.data().GetItemsInContainer();
if(!lines)              // no lines on screen
Data.Clear(); // clear all data from object( i.e. the regress data)
LongestLine = 0;  // reset for recalculation
TRect rect = GetClientRect();
height = rect.Height();
width =  rect.Width();
char str[BufLen]; // to build output string in
int coTab[NumRegrTitTab];  // for tabs for coefficient text
BOOL coeff_flag = TRUE;  // whether to draw coeff data
TEXTMETRIC metric;
tdc.SelectObject(*Font);
tdc.GetTextMetrics(metric);
HScroll.LineMagnitude = metric.tmAveCharWidth  +
metric.tmOverhang;
LineHeight = (metric.tmHeight + metric.tmExternalLeading);
if( lines )  // if there is data to paint
 {
unsigned int start, finish;   // index for range of data to output.
TCalDataIterator i(Data.data());
```

```
 TRect r;        // for drawing selected rectangles
 UINT rectStart, rectFinish;   // indexes of which rectangles to draw
TPoint Offset;
TSize TextExtent;
// calculate starting position
TPoint pos(Origin);
// modify rect to allow room to paint coefficients
rect.bottom -= 3 * LineHeight;
if(rect.Height() < 3 * LineHeight)
            {
coeff_flag = FALSE;
rect.bottom += 3 * LineHeight;
rect.Normalize();
            }
// calculate tab length
int tab[ProdReactNumTabs];
for(int j=0; j < ProdReactNumTabs; j++)
            {
if(j==0)
            {
tab[j] = (metric.tmAveCharWidth * ProdReacttabChar[j]);
continue;
            }
tab[j] = (metric.tmAveCharWidth * ProdReacttabChar[j]) + tab[j-1]; //
calculate each tab in array
            }
// draw title then modify rect to exclude this area
tdc.TabbedTextOut(
            TPoint(pos.x, 0),          // starting coordinates
            ProdReactTitleText,              // address of text
            strlen(ProdReactTitleText),        // number of characters
            sizeof(tab)/ sizeof(int),   // number of tabs in array   (if 1 all
tabs tab[0] apart)
            tab,                  // array for tab positions
            pos.x,                  // x-coord for tab expansion
ProdReactTitleExtent);
// information about size of title on the screen is put in here
LongestLine = ProdReactTitleExtent.cx;
 if(FirstTitle)   // if the title was just drawn for the first time
            {           // need to adjust local origin to account for this
Origin.y = pos.y = ProdReactTitleExtent.cy;
            }
// set pen colour to background colour of TDC object if need to
// manually erase clear parts of window
if(erase == FALSE)
            {
tdc.SelectObject( TPen(tdc.GetBkColor()) );
r.Set(pos.x + ProdReactTitleExtent.cx, rect.top, rect.right, rect.top +
ProdReactTitleExtent.cy);
tdc.Rectangle®;
            }
rect.top += ProdReactTitleExtent.cy;    // modify rectangle for presence
of title
if(pos.y < rect.top) pos.y = rect.top;
if( (rect.right < Origin.x)
|| (rect.bottom < Origin.y) ) return;  // exit function if text is offscreen
Offset = Origin - TPoint(rect.left, rect.top); // calculate offset
if( Offset.y < 0 ) // if Origin is above top of rect
            {
Offset.y = abs(Offset.y);
start = (Offset.y / LineHeight);
finish = (( rect.bottom - rect.top ) / LineHeight ) + start+1;  // length of
screen past start
            }
else
            {
start = 0;
finish = ( (rect.bottom - Origin.y) / LineHeight)+1;
            }
if(finish > lines) finish = lines;   // stay within range of data array
i.Restart(start, finish); // set iterators range
// draw text
while(i)  // for whole range to draw
            {
i++->Sprintf(str,BufLen);  // build string
tdc.TabbedTextOut(
            pos,          // starting coordinates
            str,          // address of text
            strlen(str),     // number of characters
            sizeof(tab)/ sizeof(int),  // number of tabs in array   (if 1 all
tabs tab[0] apart)
            tab,          // array for tab positions
            pos.x,          // x-coord. for tab expanding
            TextExtent);     // information about size of text on the
screen is put in here
          if(erase == FALSE)  // if rect not erased first clear off area
not written in
            {
if( (pos.x + TextExtent.cx) < width)
            {
            r.Set(pos.x + TextExtent.cx,    // end of string
                                    pos.y,          // row of text
                                    width,          // right hand edge of
client area
(pos.OffsetBy(0, LineHeight)).y); // down 1 line
tdc.Rectangle®;  // draw blank rectangle
            }
            }
pos.Offset(0, LineHeight);   // move down a line
if(TextExtent.cx > LongestLine) LongestLine = TextExtent.cx;
            }
if(erase==FALSE)
            {
if(pos.y < height)
            {
r.Set( 0,
pos.y,
width,
height);      // set rectangle to clear rest of screen if empty
tdc.Rectangle®; // draw rectangle r in background colour, already set in
loop above
            }
            }
if(coeff_flag)  // if flag is set to draw coeff data. i.e. is there room?
{  // draw coefficient text
coTab[0] =  LinRegrTitTab[0] * metric.tmAveCharWidth;        // calc
tabs

for(int o=1; o < NumRegrTitTab; o++)
            {
coTab[o] =  (LinRegrTitTab[o] * metric.tmAveCharWidth) + coTab[o-
1];
            }
pos.y = rect.bottom + LineHeight;
char *coefftit = new char[strlen(LinRegressTitle) + 20];
strcpy(coefftit, "\t");
strcat(coefftit, LinRegressTitle);
if(CalView == REACTANT) strcat(coefftit, "\tReactant");
else strcat(coefftit, "\tProduct");

tdc.TabbedTextOut(
            pos,          // starting coordinates
            coefftit,          // address of text
            strlen(coefftit),     // number of characters
sizeof(coTab)/ sizeof(int),
// number of tabs in array   (if 1 all tabs tab[0] apart)
            coTab,          // array for tab positions
            pos.x,          // x-coord. for tab expanding
            TextExtent);     // information about size of text on the
screen is put in here
LinRegress l = Data.regress();
delete coefftit;
ostrstream out(str, BufLen-1);
out << "\t" << l.m << "\t" << l.c << "\t" << l.CorrCoeff_sqr << "\t" << l.rms
<< "\t"
<< l.PerCentErrorM;
out.put('\0');
pos.Offset(0, LineHeight);   // move printing position
```

```
tdc.TabbedTextOut(
                pos,          // starting coordinates
                str,          // address of text
                strlen(str),    // number of characters
                sizeof(coTab)/ sizeof(int), // number of tabs in array   (if 1
all tabs tab[0] apart)
                coTab,        // array for tab positions
                pos.x,        // x-coord. for tab expanding
                TextExtent);    // information about size of text on the
screen is put in here
  }
 /* ***************************************************************
Rectangle drawing part
 ***************************************************************/
if(!Selected->IsEmpty())  // some records are selected
                {
rectStart = rectFinish = 0;
long numSel = Selected->GetItemsInContainer();
for(long m = 0; m < numSel; m++) // decide which selected records are
onscreen
                {
if( (*Selected)[m] <= start)
                {
rectStart = m;
break;
                }
                }
for(m = numSel - 1; m >=0; m--) // decide which selected records are
onscreen
                {
if(finish <= (*Selected)[m]) // if the index of a selected row is greater
than
{       // the number of rows of data onscreen, or if the index
rectFinish = finish;  // and the number of data rows are equal
break; // set last selected to draw as the last data row
                }
                else if( finish >= (*Selected)[m] )   // otherwise, as soon as
the index of a selected
                {       // row is that of a row onscreen
rectFinish =  m; // set last selected to draw as this index
break;
                }
                }
for(m = rectStart; m <= rectFinish; m++)
                {
r.Set(0,
Origin.y + ((*Selected)[m]) * LineHeight,
rect.Width(),
Origin.y + ((*Selected)[m] * LineHeight) + (LineHeight*1.05) );
if(!tdc.InvertRect(&r)) return;  // exit Paint if rectangle draw fails
                }
}       // end of if(!Selected->IsEmpty())
 /* ***************************************************************
End of Rectangle drawing part
 ***************************************************************/
FirstTitle = FALSE;  // the title has been drawn at least once
} // end of if(lines)
}
void TMainWindow::Modified(BOOL yes)
{
IsModified = yes;
}
void TMainWindow::DebugInfo()
{
// debugging calls to needed data
TCalData* d = (ReactCalData.data())[0];
int vert = GetScrollPos(SB_VERT);
int minpos;
int maxpos;
GetScrollRange(SB_VERT, minpos, maxpos);
KinetData* k = (*KinData)[0];
}
/*_____________________________________________ CONFIG
                FUNCTIONS
_____________________________________________*/
void TMainWindow::HandleCMConfig()
{
ConfigTrans.Set(ConfigData);
if( TConfigDialog(this, TResId(ID_CONFIG_DLG)).Execute() ==
IDOK )
{               // ok button was pressed
//  update parameters
ConfigData.Set(ConfigTrans);
KinetData::Config.Set(ConfigTrans);
Modified(TRUE);
}
}
/*_____________________________________________
DATA       FUNCTIONS
_____________________________________________*/
void TMainWindow::HandleCMDNewKin()
{
if(KinData->IsEmpty())   // for first data used set coeff's as for cal data
{
LinRegress reactRegress =  ReactCalData.regress();
LinRegress prodRegress = ProdCalData.regress();
sprintf(ReactTrans.CoeffM, "%g", reactRegress.m);
sprintf(ReactTrans.CoeffC, "%g", reactRegress.c);
sprintf(ProdTrans.CoeffM, "%g", prodRegress.m);
sprintf(ProdTrans.CoeffC, "%g", prodRegress.c);
}
CalViewMode(KINETIC);  // change to kinetic data view
if(TReactDialog(this, TResId(ID_REACT_DLG)).Execute() == IDOK)
{
KinData->Add( new KinetData(ReactTrans) );
Modified(TRUE);
FileType.NumRecsKin++;
if(TProdDialog(this, TResId(ID_PROD_DLG)).Execute() == IDOK)
                {
// set object with product data from dialog box
int n = (KinData->GetItemsInContainer() - 1);
(*KinData)[n]->Set( ProdTrans );
                }
}
UpdateScrollData();
Invalidate(FALSE);
UpdateWindow();
}
void TMainWindow::HandleNewProd()
{
CalViewMode(PRODUCT);
NewCal();
Invalidate(FALSE);
UpdateWindow();
UpdateScrollData();
}
void TMainWindow::HandleNewReact()
{
CalViewMode(REACTANT);
NewCal();
Invalidate(FALSE);
UpdateWindow();
UpdateScrollData();
}
void TMainWindow::NewCal()
{
 TCalibrationData* d;   // for data array
 long *l;      // for number of records of relevant type
TCalData* f;
if(CalView == KINETIC) CalViewMode(REACTANT); // if not then
already in one of the two
// calibration modes
if(CalView == REACTANT)
 {
d = &ReactCalData.data(); // set data reference to correct array
l = &(FileType.NumRecsReact);
 }
else
```

```
 {
CalViewMode(PRODUCT);
d = &ProdCalData.data();
l = &(FileType.NumRecsProd);
 }
//convert pointers to a references
TCalibrationData& data = *d;
long& NumRecs = *l;

if(TCalDialog(this, TResId(ID_CAL_DLG)).Execute() == IDOK)
 {
Modified(TRUE);
f = new TCalData(CalTrans);
data.Add(f );
NumRecs++;
 }
}
void TMainWindow::HandleDelete()
{
switch (CalView)
 {
case REACTANT :
            {
Delete(&(ReactCalData.data()), 0, *ReactSelected,
FileType.NumRecsReact);
break;
            }
case PRODUCT :
            {
Delete(&(ProdCalData.data()), 0, *ProdSelected,
FileType.NumRecsProd);
break;
            }
case KINETIC :
            {
Delete(0, KinData, *KinSelected, FileType.NumRecsKin);
break;
            }
default :
            {
return; // CalView corrupted
            }
 }
}
int TMainWindow::Delete(TCalibrationData* CalData, TKinData*
KData,
TLongArray& List, long& NumRecs)
 {
// must delete in reverse order otherwise as the indexes get shifted to
allow for the
// destroyed member the wrong object is deleted the next time
if( (!CalData) && (!KData) )  // nulls passed to both data's
return -1;  // have no data to use
if( (CalData) && (KData) ) // data passed to both data's
return -1;  // don't know which to use
if(CalData)  // delete calibration data
 {
TCalibrationData& data = *CalData;  // set up for use in loop
long j = List.GetItemsInContainer(); // find number of selected objects
for(long i=0; i < j; i++)
            {   // start at highest occupied indice and move to 0, if don't
start at highest
            data.Destroy( List[j-i-1] ); // deleting records changes
indices of others to
            }    // be deleted
NumRecs -= List.GetItemsInContainer();
List.Flush();
 }
else
 {
TKinData& data = *KData;
long j = List.GetItemsInContainer(); // find number of selected objects

 for(long i=0; i<j; i++)
{               // start at highest occupied indice and move to 0
data.Destroy( List[j-i-1] );
            }
NumRecs -= List.GetItemsInContainer();
List.Flush();
 }
UpdateScrollData();
Modified(TRUE);
Invalidate();
UpdateWindow();
}
void TMainWindow::HandleCMDEdit()
 {
switch (CalView)
 {
case REACTANT :
            {
Edit(&(ReactCalData.data()), 0, *ReactSelected,
FileType.NumRecsReact);
break;
            }
case PRODUCT :
            {
Edit(&(ProdCalData.data()), 0, *ProdSelected,
FileType.NumRecsProd);
break;
            }
case KINETIC :
            {
Edit(0, KinData, *KinSelected, FileType.NumRecsKin);
break;
            }
default :
            {
return; // CalView corrupted
            }
 }
}
int TMainWindow::Edit(TCalibrationData* CalData, TKinData* KData,
TLongArray& List, long& NumRecs)
 {
if( (!CalData) && (!KData) )  // nulls passed to both data's
return -1; // have no data to use
if( (CalData) && (KData) ) // data passed to both data's
return -1;  // don't know which to use
if(CalData)    // need to edit caldata
 {
TLongArrayIterator i(List);
while(i)
            {
CalTrans.Set( ((*CalData)[i.Current()])->data() );
if(TCalDialog(this, TResId(ID_CAL_DLG)).Execute() == IDOK)
            {
(*CalData)[i++]->Set(CalTrans);
            }
else break;  // otherwise don't bother
            }
 }
else  // kinetic data
 {
TLongArrayIterator i(List);
while(i)
            {
ReactTrans.Set( ((*KData)[i.Current()])->data()  );
if(TReactDialog(this, TResId(ID_REACT_DLG)).Execute() == IDOK)
            {
(*KData)[i.Current()]->Set(ReactTrans);
ProdTrans.Set( ((*KData)[i.Current()])->data() );
if(TProdDialog(this, TResId(ID_PROD_DLG)).Execute() == IDOK)
(*KData)[i++]->Set(ProdTrans);
else break;  // otherwise forget it
            }
else break;  // otherwise don't bother
            }
```

```
} // end of outer else
if(!List.IsEmpty()) // if there were records selected
        {
Modified(TRUE);
Invalidate();
UpdateWindow();
        }
} // end of Edit()
/*-------------------------------------------------
USER INTERFACE    FUNCTIONS
-------------------------------------------------*/
void TMainWindow::EvLButtonDblClk(UINT Key, TPoint& p)
{
 BOOL IsSelected = TRUE;   // are any records in this view selected
 BOOL edit = TRUE;       // should the edit part be performed
 switch (CalView)      // check each case for selected records
 {
case REACTANT :
                {
if(ReactSelected->IsEmpty() ) IsSelected = FALSE;
break;
                }
case PRODUCT :
                {
if(ProdSelected->IsEmpty() ) IsSelected = FALSE;
break;
                }
case KINETIC :
                {
if(KinSelected->IsEmpty() ) IsSelected = FALSE;
break;
                }
default :
                {
return; // CalView has been corrupted somehow
                }
} // end of switch
if(!IsSelected) // no records selected
 {
EvLButtonDown(Key, p); // can do EvLButtonDown function
 }
else
 {
 TLongArray *Selected;        // for selected list
 TPoint *point;           // for origin
long lines;
long LineNo;
switch (CalView)
        {
case REACTANT :
                {
lines = ReactCalData.data().GetItemsInContainer();
Selected = ReactSelected;
point = &ReactLocOrigin;
break;
                }
case PRODUCT :
                {
lines = ProdCalData.data().GetItemsInContainer();
Selected = ProdSelected;
point = &ProdLocOrigin;
break;
                }
 case KINETIC :
                {
lines = KinData->GetItemsInContainer();
Selected = KinSelected;
point = &KineticLocOrigin;
break;
                }
 default :
                {
 return; // CalView has been corrupted somehow
                }
} // end of switch
TPoint& origin = *point;       // set up references for the relevant
variables
TPoint Offset(p-origin);
// debugging part
long sel[400];
for(int h=0; h < Selected->GetItemsInContainer(); h++)
        {
sel[h] = (*Selected)[h];
        }
// end of debugging insert
if(lines)  // if there is data
        {
if( (Offset.y < 0) || (Offset.y > (lines*LineHeight)) ) // outside text area
        {
Invalidate(FALSE);       // refresh window
UpdateWindow();
return; // exit function
        }
else
        {
LineNo = (Offset.y/LineHeight);
if(!Selected->HasMember(LineNo)) // if index is not already on
Selected list
        {
edit = FALSE;       // do not call edit
        }
        }
} // end of if (lines)
} // end of else
if(edit) HandleCMDEdit();   // if ok to edit
}
void TMainWindow::EvLButtonDown(UINT Key, TPoint& p)
{
 TLongArray *Selected;        // for selected list
 TPoint *point;           // for origin
long lines;
long LineNo; // for temporary use before assignation to relevant long
variable
static long KinLineNo, ReactLineNo, ProdLineNo;
// to be kept from one call to the next
switch (CalView)
 {
case REACTANT :
                {
LineNo = ReactLineNo;
lines = ReactCalData.data().GetItemsInContainer();
Selected = ReactSelected;
point = &ReactLocOrigin;
break;
                }
case PRODUCT :
                {
LineNo = ProdLineNo;
lines = ProdCalData.data().GetItemsInContainer();
Selected = ProdSelected;
point = &ProdLocOrigin;
break;
                }
case KINETIC :
                {
LineNo = KinLineNo;
lines = KinData->GetItemsInContainer();
Selected = KinSelected;
point = &KineticLocOrigin;
break;
                }
default :
                {
return; // CalView has been corrupted somehow
                }
 }
 TPoint& origin = *point;       // set up references for the relevant
variables
```

```
TPoint Offset(p-origin);
// debugging part
long sel[400];
for(int h=0; h < Selected->GetItemsInContainer(); h++)
 {
sel[h] = (*Selected)[h];
 }
// end of debugging insert
if(lines)  // if there is data
 {
if(Key != LKEY_SHIFT) // shift key is not being held down
            {
if( (Offset.y < 0) || (Offset.y > (lines*LineHeight)) ) // outside text area
            {
if(!Selected->IsEmpty() ) Selected->Flush();
Invalidate(FALSE);
UpdateWindow();
return; // exit function
            }
else if(Key == LKEY_CONTROL)  // control key is pressed
            {
LineNo = (Offset.y/LineHeight);
if(Selected->HasMember(LineNo)) // if index is already on Selected list
                        Selected->Destroy(LineNo);  // remove it
            else Selected->Add(LineNo);   // otherwise add it
            }
else     // else control key has not been pressed
            {
LineNo = (Offset.y/LineHeight);
if(Selected->HasMember(LineNo)) // if index is already on Selected list
Selected->Flush(); // remove all Selected
else
            {
Selected->Flush(); // remove all Selected
Selected->Add(LineNo);   // otherwise add it
            }
            }
else  // shift key is down
            {
if( !Selected->IsEmpty() )  // some records are already Selected
            {
long Min, Max;
long Old = LineNo;  // record Selected the last time this function was
called
LineNo = (Offset.y/LineHeight);
            if(Offset.y < 0)        // outside text area
                        LineNo = 0;     // set to first line
else if(Offset.y > (lines*LineHeight)) // outside text area
LineNo = KinData->GetItemsInContainer();
Min = min(Old, LineNo);
// should be faster to allocate the range to the Selected
Max = max(Old, LineNo); // list if it is done in the correct order (Less
sorting)
for(long i= Min; i <=Max; i++)
            {
if(!Selected->HasMember(i) )
Selected->Add(i);
            }
}  // end of if(Sele...
else // there are no records Selected
            {
LineNo = (Offset.y/LineHeight);
Selected->Add(LineNo);
            }
} // end of else (shift)
switch (CalView)
 {
case REACTANT :
                        {
ReactLineNo = LineNo;
break;
                        }
case PRODUCT :
                        {
ProdLineNo = LineNo;
break;
                        }
case KINETIC :
                        {
KinLineNo = LineNo;
break;
                        }
default :
                        {
return;  // CalView has been corrupted somehow
                        }
 }
// debugging part
for(int h=0; h < Selected->GetItemsInContainer(); h++)
 {
sel[h] = (*Selected)[h];
 }
// end of debugging insert
Invalidate(FALSE);
UpdateWindow();
} // end of if (lines)
}
void TMainWindow::UpdateScrollData(BOOL SetupCall)
{
TRect rTemp = GetClientRect();
int lines;
TScrollData* H, *V;  // to allow choosing of relevant scroll data
int linelength;     // holds relevant longest line value
if(SetupCall == TRUE)
 {
InitialiseScroll(KineticHScroll, 15, 300);    // initialise scroll data
InitialiseScroll(KineticVScroll, 15, 100);
InitialiseScroll(ReactHScroll, 15, 300);
InitialiseScroll(ReactVScroll, 15, 300);
InitialiseScroll(ProdHScroll, 15, 300);
InitialiseScroll(ProdVScroll, 15, 300);
SetScrollRange(SB_HORZ, KineticHScroll.LowValue,
KineticHScroll.HighValue, FALSE);
SetScrollRange(SB_VERT, KineticVScroll.LowValue,
KineticVScroll.HighValue, FALSE);
SetScrollPos(SB_HORZ, KineticHScroll.Position);
SetScrollPos(SB_VERT, KineticVScroll.Position);
return;
 }
switch (CalView)
 {
case KINETIC: {
H = &KineticHScroll;
V = &KineticVScroll;
linelength = LongestLineKin;
lines = KinData->GetItemsInContainer();
break;
                                                }
case REACTANT: {
H = &ReactHScroll;
V = &ReactVScroll;
linelength = LongestLineReact;
lines = ReactCalData.data().GetItemsInContainer();
break;
                                                }
case PRODUCT: {
H = &ProdHScroll;
V = &ProdVScroll;
linelength = LongestLineProd;
lines = ProdCalData.data().GetItemsInContainer();
break;
                                                }
} // end of switch
TScrollData& HScroll = *H; // change pointers to references (rest of
function already
TScrollData& VScroll = *V; // in reference form not pointer form
```

```
 VScroll.LineMagnitude = LineHeight; // HScroll.LineMagnitude is
 assigned in Paint()
 VScroll.PageMagnitude = rTemp.Height();
 HScroll.PageMagnitude = rTemp.Width();
 //GetClientRectreturns a TRect. can then call Height() funtion to return
 height
 if(!VScroll.PageMagnitude) VScroll.PageMagnitude = 1;
 if(!HScroll.PageMagnitude) HScroll.PageMagnitude = 1;
 // to stop divide by zero errors
 VScroll.LowValue = 0;
 VScroll.HighValue = lines;
 // cast to double to cause sum to give a floating point number
 if( (VScroll.NumPages = (lines*(double)LineHeight) /
 VScroll.PageMagnitude ) < 1)
 VScroll.NumPages = 1;
 HScroll.HighValue = linelength/HScroll.LineMagnitude;
 // longest line in logical units now not characters
 if( (HScroll.NumPages =
 // cast to double to cause sum to give a floating point number
 (double)linelength / HScroll.PageMagnitude)
                                 < 1)  HScroll.NumPages = 1;
 SetScrollRange(SB_HORZ, HScroll.LowValue, HScroll.HighValue,
 FALSE);
 SetScrollRange(SB_VERT, VScroll.LowValue, VScroll.HighValue,
 FALSE);
 SetScrollPos(SB_HORZ, HScroll.Position);
 SetScrollPos(SB_VERT, VScroll.Position);
 } // end of UpdateScrollData()
void TMainWindow::EvHScroll(UINT scrollcode, UINT thumbpos,
HWND )
 {
 TScrollData* H, *V; // to allow choosing of relevant scroll data
 TPoint *O;        // to hold relevant origin data
 switch (CalView)
  {
 case KINETIC: {
 H = &KineticHScroll;
 V = &KineticVScroll;
 O = &KineticLocOrigin;
 break;
                                                  }
 case REACTANT: {
 H = &ReactHScroll;
 V = &ReactVScroll;
 O = &ReactLocOrigin;
 break;
                                                  }
 case PRODUCT: {
 H = &ProdHScroll;
 V = &ProdVScroll;
 O = &ProdLocOrigin;
 break;
                                                  }
 } // end of switch
 TScrollData& HScroll = *H; // change pointers to references (rest of
 function already
 TScrollData& VScroll = *V; // in reference form not pointer form
 TPoint& origin = *O;     // set reference to origin (so that it can be
 changed)
 switch(scrollcode)
  {
 case SB_THUMBPOSITION: {  // absolute thumb move (4)
 HScroll.Position = thumbpos;
 SetScrollPos(SB_HORZ, HScroll.Position);
 origin.x = (-(HScroll.Position * HScroll.LineMagnitude));
 Invalidate(FALSE);
 UpdateWindow();
 break;

                                 }
 case SB_THUMBTRACK:   { // drag thumb to specified position (5)
 HScroll.Position = thumbpos;
 SetScrollPos(SB_HORZ, HScroll.Position);
 origin.x = -(HScroll.Position * HScroll.LineMagnitude);
 Invalidate(FALSE);
 UpdateWindow();
 break;

                                 }
 /* case SB_ENDSCROLL:   { // end of scroll operation (8)
 MessageBox("End Of Scroll", "Information", MB_OK);
 break;

                                 } */
 case SB_PAGELEFT:   { // SCROLL 1 PAGE TO THE LEFT (2)
 HScroll.Position -= HScroll.Range() / HScroll.NumPages;
 if(HScroll.Position < HScroll.LowValue )
 HScroll.Position = HScroll.LowValue;
 SetScrollPos(SB_HORZ, HScroll.Position);
 origin.x = -(HScroll.Position * HScroll.LineMagnitude);
 Invalidate(FALSE);
 UpdateWindow();
 break;

                                 }
 case SB_PAGERIGHT:   { // SCROLL 1 PAGE TO THE RIGHT (3)
 HScroll.Position += HScroll.Range() / HScroll.NumPages;
 if(HScroll.Position > HScroll.HighValue )
 HScroll.Position = HScroll.HighValue;
 SetScrollPos(SB_HORZ, HScroll.Position);
 origin.x = -(HScroll.Position * HScroll.LineMagnitude);
 Invalidate(FALSE);
 UpdateWindow();
 break;

                                 }
 case SB_LINERIGHT:   { // SCROLL 1 LINE TO THE RIGHT (1)
 HScroll.Position++;
 if(HScroll.Position > HScroll.HighValue )
 HScroll.Position = HScroll.HighValue;
 SetScrollPos(SB_HORZ, HScroll.Position);
 origin.x = -(HScroll.Position * HScroll.LineMagnitude);
 Invalidate(FALSE);
 UpdateWindow();
 break;

                                 }
 case SB_LINELEFT:   { // SCROLL 1 LINE TO THE LEFT (0)
 HScroll.Position--;
 if(HScroll.Position < HScroll.LowValue )
 HScroll.Position = HScroll.LowValue;
 SetScrollPos(SB_HORZ, HScroll.Position);
 origin.x = -(HScroll.Position * HScroll.LineMagnitude);
 Invalidate(FALSE);
 UpdateWindow();
 break;

                                 }
 case SB_RIGHT:   { // end of scroll operation (7)
 MessageBox("SB_RIGHT", "Information", MB_OK);
 break;
                 }
 case SB_LEFT:   { // end of scroll operation (6)
 MessageBox("SB_LEFT", "Information", MB_OK);
 break;
                 }
 } // end of switch
  }
 void TMainWindow::EvVScroll(UINT scrollcode, UINT thumbpos,
 HWND )
 {
 TScrollData* H, *V; // to allow choosing of relevant scroll data
  TPoint *O;        // to hold relevant origin data
  TSize* S;         // to hold the relevant title extent
 switch (CalView)
  {
 case KINETIC: {
 H = &KineticHScroll;
```

```
V = &KineticVScroll;
O = &KineticLocOrigin;
S = &KineticTitleExtent;
break;
                                                    }
case REACTANT: {
H = &ReactHScroll;
V = &ReactVScroll;
O = &ReactLocOrigin;
S = &ProdReactTitleExtent;
break;
                                                    }
case PRODUCT: {
H = &ProdHScroll;
V = &ProdVScroll;
O = &ProdLocOrigin;
S = &ProdReactTitleExtent;
break;
                                                    }
} // end of switch
TScrollData& HScroll = *H; // change pointers to references (rest of
function already
TScrollData& VScroll = *V; // in reference form not pointer form
 TPoint& origin = *O;     // set reference to origin
 TSize& titleextent = *S;  // set reference to title extent
switch(scrollcode)
 {
case SB_THUMBPOSITION: {  // absolute thumb move (4)
VScroll.Position = thumbpos;
SetScrollPos(SB_VERT, VScroll.Position);
origin.y = (-(VScroll.Position * VScroll.LineMagnitude))
titleextent.cy; // + title height
Invalidate(FALSE);
UpdateWindow();
break;

                                    }
 case SB_THUMBTRACK:   { // drag thumb to specified position (5)
                                    VScroll.Position = thumbpos;
                          SetScrollPos(SB_VERT, VScroll.Position);
origin.y = -(VScroll.Position * VScroll.LineMagnitude)
titleextent.cy; // + title height
Invalidate(FALSE);
UpdateWindow();
break;

                                    }
/* case SB_ENDSCROLL:   { // end of scroll operation (8)
MessageBox("End Of Scroll", "Information", MB_OK);
break;

                                    } */
case SB_PAGEUP:   { // SCROLL 1 PAGE UP (2)
VScroll.Position -= VScroll.Range() / VScroll.NumPages;
if(VScroll.Position < VScroll.LowValue )
VScroll.Position = VScroll.LowValue;
SetScrollPos(SB_VERT, VScroll.Position);
origin.y = -(VScroll.Position * VScroll.LineMagnitude)
titleextent.cy; // + title height;
Invalidate(FALSE);
UpdateWindow();
break;

                                    }
 case SB_PAGEDOWN:   { // SCROLL 1 PAGE DOWN(3)
 VScroll.Position += VScroll.Range() / VScroll.NumPages;
 if(VScroll.Position > VScroll.HighValue )
 VScroll.Position = VScroll.HighValue;
 SetScrollPos(SB_VERT, VScroll.Position);
 origin.y = -(VScroll.Position * VScroll.LineMagnitude)
 titleextent.cy; // + title height;
 Invalidate(FALSE);
 UpdateWindow();
 break;
                                    }
case SB_LINEDOWN:   { // SCROLL 1 LINE DOWN (1)
VScroll.Position++;
if(VScroll.Position > VScroll.HighValue )
VScroll.Position = VScroll.HighValue;
SetScrollPos(SB_VERT, VScroll.Position);
origin.y = -(VScroll.Position * VScroll.LineMagnitude)
titleextent.cy; // + title height;
Invalidate(FALSE);
UpdateWindow();
break;

                                    }
case SB_LINEUP:   { // SCROLL 1 LINE UP (0)
VScroll.Position--;
if(VScroll.Position < VScroll.LowValue )
VScroll.Position = VScroll.LowValue;
SetScrollPos(SB_VERT, VScroll.Position);
origin.y = -(VScroll.Position * VScroll.LineMagnitude)
titleextent.cy; // + title height;
Invalidate(FALSE);
UpdateWindow();
break;

                                    }
case SB_BOTTOM:   { // end of scroll operation (7)
MessageBox("SB_RIGHT", "Information", MB_OK);
break;
                                                    }
case SB_TOP:   { // end of scroll operation (6)
MessageBox("SB_LEFT", "Information", MB_OK);
break;
                                    }
} // end of switch
}
void TMainWindow::EvRButtonDown(UINT key, TPoint& p)
{
if(CanClose())
Close();
}
BOOL TMainWindow::CanClose()
{
if(IsModified)
 {
long ans = MessageBox("Document has been changed.  Want to
save changes?",
"Query", MB_YESNOCANCEL | MB_ICONQUESTION);
if(ans == IDYES)   // user wants to save changes
          {
HandleCMFileSaveAs(); // if want to save file, save it
return TRUE;    // ok to save now
          }
else if(ans==IDNO) return TRUE;  // don't want to save changes; ok to
close
else return FALSE;      // don't want to close document
 }
return TRUE;  // default if message box returned something other than
expected
}
/*_____________________________________
TWinApp class
for application object
_____________________________________*/
TWinApp::~TWinApp()
{
TApplication::~TApplication();
if(MainWindow)
delete MainWindow;
}
void TWinApp::InitMainWindow()
{
MainWindow = new TFrameWindow(0,
"Kinetic Calc (v.1)",
```

```
new TMainWindow);
// load DLL's
EnableBWCC();
// load the menu resource
MainWindow->AssignMenu(TResId(ID_KINET_MENU));
MainWindow->Attr.AccelTable = TResId(ID_KINET_MENU);
MainWindow->SetIcon(this, KINET1_ICON);
}


int OwlMain(int /* argc */, char** /* argv[] */)
{
TWinApp app;
return app.Run();
}
```

Kinet1.def

```
NAME        Kinet1
DESCRIPTION  'An OWL Windows Application'
EXETYPE     WINDOWS
CODE  PRELOAD MOVEABLE DISCARDABLE
DATA  PRELOAD MOVEABLE MULTIPLE
HEAPSIZE              1024
STACKSIZE             1638
```

References

1 *Introduction to Organic Chemistry*. 3rd Ed. by A. Streitwieser and Jr. Clayton, H. Heathcock, Colier Macmillan 1989.

2 *Agrochemicals: Preparation and Mode of Action*. by R. J. Cremlyn, Wiley (New York) 1991.
(a) p. 82 (b) p. 83 (c) p. 352 (d) p. 345

3 G. W. Woodwell, P. R. Craig, H. A. Johnson; *Science*, 1971, **174**, 1101.

4 *Soil Processes* by S. Ross. Routledge (London) 1989.

5 *Nature and Origins of Pollution of Aquatic Systems by Pesticides* by C. A. Edwards in *Pesticides in Aquatic Environments* by M. A. Q. Khan. Plenum Press (New York) 1977.

6 *Pollutants and Animals. A Factual Perspective* by F. Moriarty, George Allen and Unwin Ltd (London) 1975, p. 22.

7 *Pesticides and the Reproduction of Birds* in *Chemistry in the Environment*, by Peakall, *Readings from* ***Scientific American***. Introduction by C. Hamilton, W. H. Freeman and Company 1973, pp. 117-119.

8 *Insecticide Biochemistry and Physiology* by C. F. Wilkinson (ed), Heyden (London) 1976.

9 *Marine Pollution*, 2nd Ed. by R. B. Clark., Clarendon Press 1989.

10 L. C. Dickson, S. C. Buzik; *Vet. Hum. Toxicol.*, 1993, **35**(1), 68.

11 E. Johnson; *Toxicology*, 1992, **21**(6), 451.

12 A. H. Smith, D. G. Patterson Jr., M. L. Warner, R. MacKenzie, L. L. Needman; *Journal of the National Cancer Institute*, 1992, **84**(2), 104.

13 *Dioxin : toxicological and chemical aspects*, Edited by Flaminio Cattabeni, Aldo Cavallaro, Giovanni Galli, SP Medical and Scientific Books (New York) 1978.

14 O. Hutzinger, M. Blumich; *Chemosphere*, 1985, **14**(6-7), 581.

15 M. Gonzalez, B. Jimenez, M. Fernandez, L. Hernandez, *Toxicological and Environ. Chem.*, 1991, **33**, 169.

16 M. Erickson, C. Cole, J. Flora, P. Gorman, C. Haile, G. Hinshaw, F. Hopkins, S. Swanson, D. Heggem; *Chemosphere*, 1985, **14**(6-7), 855.

17 R. E. Clement, H. M. Tosine, B. Ali; *Chemosphere*, 1985, **14**(6-7), 815.

18 H. Muto, K. Saitoh, Y. Takizawa; *Bull. Environ. Contam. Toxicol.*, 1993, **50**, 340.

19 A. Kocan, V. Bencko, W. Sixl; *Toxicological and Environmental Chemistry*, 1992, **36**, 33.

20 R. Konduri, E. Altwicker; *Chemosphere*, 1994, **28**(1), 23.

21 D. Tirey, P. Taylor, J. Kasner, B. Dellinger; *Combust. Sci. and Tech.*, 1990, **74**, 137.

22 E. Altwicker, R. Konduri, C. Lin, M. Milligan; *Combust. Sci. and Tech.*, 1993, **88**, 349.

23 A. Blankenship, D. Chang, A. Jones, P. Kelly, I. Kennedy, F. Matsumara, R. Pasek; *Chemosphere*, 1994, **28**(1), 183.

24 L. Stieglitz, G. Zwick, J. Beck, W. Roth, H. Vogg; *Chemosphere*, 1989, **18**(1-6), 1219.

25 V. Boscak; *Waste Management and Research*, 1993, **11**, 493.

26 K. Naikwadi, I. Albrecht, F. Karasek; *Chemosphere*, 1993, **27**(1-3), 335.

27 C. Rittmeyer, J. Vehlow; *Chemosphere*, 1993, **26**(12), 2129.

28 European Community Directive Cm 2181 (1993).

29 *The Private Sector Waste Contracting Industry In Britain, Part 1*; ETSU B1238, Aspinwall & Company Ltd. 1990.

30 *Waste Incineration: BPEO?* M. Pennington, D. Symon and V. Carter, 1995, unpublished.

31 J. Glass, C.N. Hinshelwood; *J. Chem. Soc.*, 1929, 1815.

32 A. Maccoll; *Chem. Rev.*, 1969, 33.

33 J. Lubkowski, T. Janiak, J. Czerminski, J. Blazejowski; *Thermochimica Acta*, 1989, **155**, 7.

34 N. Chopra, J. J. Domanski; *Beitrage zur Tabakforschung*, 1972, **6**(3), 139.

35 N. Chopra, J. Domanski, N. Osborne; *Beitrage zur Tabakforschung*, 1970, **5**(4), 167.

36 N. Chopra, L. Sherman; *Anal. Chem.*, 1972, **44**(6), 1036.

37 N. Chopra, J. Thekkekandan; *Beitrage zur Tabakforschung*, 1973, **7**(2), 88.

38 W. C. Herndon; *Journal of Chemical Education*, 1964, **41**(8), 425.

39 *The Foundations of Chemical Kinetics*, by S. W. Benson, McGraw-Hill Book Co. (New York) 1960, pp. 61-63.

40 *Gas Kinetics : studies in modern chemistry*, by M. F. Mulcahy, Thomas Nelson and Sons Ltd. 1973, p. 52.

41 M. Mulcahy, D. Williams; *Aust. J. Chem.*, 1961, **14**, 534.

42 I. Davidson, A. Baldwin, A. Howard; *J. Chem. Soc., Faraday Trans I*, 1975, **71**, 972.

43 I. Davidson, G. Eaton, K. Hughes; *Journal of Organometallic Chemistry*, 1988, **347**, 17.

44 R. Gordon, J. Szita, E. Foeder; *Anal. Chem.*, 1982, **54**, 478.

45 D. Barton, P. Onyon; *J. Am. Chem. Soc.*, 1950, **72**, 988.

46 D. Barton; *J. Chem. Soc.*, 1949, 148.

47 *Chemical Kinetics. A Modern Survey of Gas Reactions*. by J. Nicholas, Harper and Row (London) 1976. a) p. 124, b) p. 130.

48 J. Pola; *Int. J. Chem. Kinet.*, 1983, **15**, 1119.

49 *Comprehensive Chemical Kinetics* **5**, by E. Swinbourne, C. Bamford (ed) and C. Tipper (ed), Elsevier, Amsterdam 1972.

50 R. Zitter, D. Koster, A. Cantoni, J. Pleil; *Chem. Phys.*, 1980, **46**, 107.

51 G. Huybrechts, J. Katihabwa, G. Martens, M. Nejszaten, J. Olbregts; *Bull. Soc. Chim. Belges*, 1972, **81**, 65.

52 G. Huybrechts, Y. Hubin, B. Van Mele; *Int. J. Chem. Kinet.*, 1989, **21**, 575.

53 M. Thomson, D. Lucas, C. Koshland, R. Sawyer, Y. Wu, J. Bozzelli ; *Combustion and Flame*, 1994, **98**(1-2), 155.

54 P. Taylor, D. Tirey, W. Rubey, B. Dellinger; *Combustion Science and Technology*, 1994, **101**(1-6), 75.

55 D. Seyferth, R. Damrauer, S. Andrews, S. Washburne; *J. Am. Chem. Soc.*, 1971, **93**, 3709.

56 R. Schwartz, G. Pietsch; *Z. anorg. allg. Chem.*, 1937, **232**, 249.

57 O. Nejedov, M. Manakov; *Angew. Chem. internat. Eng. ed.*, 1966, **5**(12), 1021.

58 R. Conlin, D. Wood; *J. Am. Chem. Soc.*, 1981, **103**, 1843.

59 M. Flowers, L. Gusel'nikov; *J. Chem. Soc (B)*, 1968, 419.

60 M. Flowers, L. Gusel'nikov; *J. Chem. Soc., Chem. Comm.*, 1967, 864.

61 F. O. Rice, K. F. Herzfeld; *J. Phys. & Colloid Chem.*, 1951, **55**, 975.

62 *Gas chromatography* by J. Willett, D. Kealey (Ed.), John Wiley & Sons (1987), pp. 34-42.