A Practical Investigation of Meteor-Burst Communications

by

Stuart William Melville

Submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy

in the Department of Computer Science,

University of Natal.

Durban
November 1991

Preface

The work described in this thesis was carried out in the Department of Computer Science, University of Natal, Durban, under the supervision of Professor Alan Sartori-Angus and Doctor Ortrud Oellerman. Initial work has been done since 1987, however the majority of the work presented here was carried out in the period September, 1989 to November, 1991.

These studies represent original work by the author and have not been submitted in any form to any other university. Where use has been made of the work of others, it has been duly acknowledged in the text.

Acknowledgements

First and foremost, my thanks go to James Larsen, who introduced me to the field of meteor-burst communications in 1987. Over the years his support has been invaluable.

My thanks to my supervisors, Alan Sartori-Angus and Ortrud Oellerman, for their contributions. Particular thanks go to Ortrud for her hard work and many valuable suggestions concerning the write-up of this thesis.

The work was funded by Salbu (Pty) Ltd, the FRD, and the University of Natal. My sincere thanks go to these organisations for their vital financial support, as well as for grants enabling me to attend conferences both locally and in the United States.

Salbu (Pty) Ltd of Pretoria, and Meteor Communications Corporation of Seattle, United States, both provided valuable information. Particular thanks go to Dave Larsen and Peter Handley in this regard.

A number of teams have been involved in various aspects of this work. The following people all share in this thesis :

Trail Classification - Rob Letschert, James Larsen and Wayne Goddard all played vital roles. Thanks also to our operators, Andrew Deighton, Lindsay Pratt, Debbie Sweby and Alan Gaffin.

Throughput Capacity - Robert Mawrey, James Larsen and Rob Letschert made vital contributions.

Waiting Time - James Larsen gave important advice and assistance.

Networks - David Carson made some valuable contributions.

Robyn Laing, Ortrud Oellerman, Celia Thomson, and David Fraser all gave willingly of their time in proofreading this thesis. I owe them my sincere thanks for their hard work, any mistakes that remain are my own responsibility.

Finally, my thanks and my love to the people who have provided my most vital support over the past years - my mother Wendy Hales, my brother Darryl Vine, and my constant companion Celia Thomson.

Abstract

This study considers the meteor-burst communication (MBC) environment at three levels. At the lowest level, the trails themselves are studied and analysed. Then individual links are studied in order to determine the data throughput and wait time that might be expected at various data rates. Finally, at the top level, MBC networks are studied in order to provide information on the effects of routing strategies, topologies, and connectivity in such networks.

A significant amount of theoretical work has been done in the classification of meteor trails, and the analysis of the throughput potential of the channel. At the same time the issues of wait time on MBC links, and MBC network strategies, have been largely ignored. The work presented here is based on data captured on actual monitoring links, and is intended to provide both an observational comparison to theoretical predictions in the well-researched areas, and a source of base information for the others.

Chapter 1 of this thesis gives an overview of the field of meteor-burst communications. Prior work in the field is discussed, as are the advantages and disadvantages of the channel, and current application areas.

Chapter 2 describes work done on the classification of observed meteor trails into distinctive 'families'. The rule-based system designed for this task is discussed as well as the eventual classification schema produced, which is far more comprehensive and consistent than previously proposed schemas.

Chapter 3 deals with the throughput potential of the channel, based on the observed trails. A comparison to predicted results, both as regards fixed and adaptive data-rates, is made with some notable differences between predicted

results and observed results highlighted. The trail families with the largest contribution to the throughput capacity of the channel are identified.

Chapter 4 deals with wait time in meteor-burst communications. The data rates at which wait time is minimised in the links used are found, and compared to the rates at which throughput was optimised. These are found to be very different, as indeed are the contributions of the various trail families at these rates.

Chapter 5 describes a software system designed to analyse the effect of routing strategies in MBC networks, and presents initial results derived from this system. Certain features of the channel, in particular its sporadic nature, are shown to have significant effects on network performance.

Chapter 6 continues the presentation of network results, specifically concentrating on the effect of topologies and connectivity within MBC networks.

Chapter 7 concludes the thesis, highlighting suggested areas for further research as well as summarising the more important results presented.

List of Contributions

As indicated earlier, a number of teams have been involved with various aspects of this work. In order to both outline the contributions made by the different individuals involved, and to ensure that the work done by the author is clearly outlined, the following list of contributions is presented. Clearly in any situation involving teamwork there will naturally be a flow of information and advice amongst individuals - the assignment of a particular contribution to a particular individual should thus be taken to mean that the individual was the **main** contributor to the work.

General :

The meteor-burst monitoring systems were configured, installed and operated by members of the Electronic Engineering Department, University of Natal, Durban.

Chapter 1 - Introduction :

Entirely by S W Melville.

Chapter 2 - Trail Classification :

Suggestion to research : J D Larsen (who originally envisaged 4 or 5 types, primarily underdende/overdense).

Literature research on prior classifications : S W Melville.

Decision to classify by a rule-based expert system : S W Melville

Design of expert system shell : S W Melville.

Design and implementation of system modules (learner, high-level modifier, condition-checking, query, help and lister modules) : S W Melville.

Screen displays and data file I/O : R Y Letschert.

Approximately 75 of the 83 condition routines : S W Melville.

Approximately 8 of the 83 condition routines : R Y Letschert and W D Goddard.

Statistics software : S W Melville.

Trail Identification : Chiefly S W Melville, although the entire team got involved.

Rule design : S W Melville.

Result analysis : S W Melville and J D Larsen.

Chapter 3 - Throughput Capacity

Suggestion to research : J D Larsen.

(1) "No overhead" simulation (based on formula (1))

Most theory (including suggestion that formula (1) could be applicable) : R S Mawrey.

Most algorithm design and implementation : S W Melville.

Results analysis : S W Melville and R S Mawrey.

(2) "Real-world" simulation

Algorithm design : S W Melville, R Y Letschert and J D Larsen.

Search technique : S W Melville and R Y Letschert.

Results analysis : S W Melville.

(3) Trail type Contributions.

All by S W Melville.

Chapter 4 - Wait time

Suggestion to research : J D Larsen.

Method, algorithms, analysis of results : S W Melville.

Chapter 5 - Networks I

Entire content : S W Melville.

Chapter 6 - Networks II

Generation of networks (data files) : D I Carson and S W Melville.

Method, algorithms, analysis : S W Melville.

Chapter 7 : Conclusion

Written entirely by S W Melville, clearly based largely on the other chapters with their relevant contributions from the various individuals.

Table of Contents

Appendices

1 Introduction

1.1 Overview of Meteor-Burst Communications

Every day billions of meteoroids, in orbit around the sun, collide with the earth's atmosphere. At this stage they become meteors, and typically burn up at heights ranging between 80 and 120 km above the earth's surface. Despite the relatively small size of the majority of these meteors - comparable to grains of sand - their high speed of around 35 km/sec causes ionised columns up to tens of kilometres long to be formed as they burn up under atmospheric friction. These columns reflect very high frequency radio signals beyond line of sight up to a maximum distance of some 2000 km, limited by the curvature of the earth and height of ionisation. The duration of the reflection is limited by the diffusion time of the ionised trail, but is sufficiently long to support burst mode data communication.

A connection between meteors and radio reflections was first postulated by Nagaoka in 1929 [1]. His initial premise that these meteors would be impediments to radio communication was questioned by Pickard in 1931 [2], and Skellet in 1932 [3]. Skellett identified meteor ionisation columns as phenomena that could enable enhanced radio reflection to occur at very high frequencies.

A period of intense interest in meteor communications in the late 1940's and 1950's resulted in the development of a great deal of current theory. Lovell and Clegg [4] did important work in establishing electron densities of meteor trails. Lovell [5] further established that sporadic meteors were members of the solar system and did not come from interstellar space, a point that was in great dispute at the time. McKinley and Millman [6] did vital early work in establishing basic theory as well as investigating shower effects, and were the first to suggest that high-altitude winds could have a significant effect on meteor trails. Further work by McKinley [7] dealt with meteor

velocities, and provided important data on sporadic meteors. Work prior to this had tended to centre on shower meteors, due to the body of available theory on these, their predictability, and their generally greater visibility. (Early work was largely done by comparing radio signals with actual visual observations of meteors.) Current systems rely on the daily arrival of sporadic meteors rather than the annual periods of meteors occurring in such showers as the Geminids or Quadrantids, so this work was of great importance. Hawkins[8] made further valuable contributions to the study of sporadic meteors and, together with Brown, made the first attempt at a comprehensive study of meteor trail reflection characteristics [9].

The early back-scatter experiments finally led to the development of the first practical forward-scatter experiment, and indeed, the first meteor-burst link with the Canadian Janet system. Operational in March 1954, the system employed double side-band AM, and has been described in [10, 11]. The optimal frequency for the system was determined to be in the 30-50 MHz range, and it had an average performance cited at thirty-four words per minute [10]. Important discoveries made as a result of JANET's operation were that performance would vary considerably in line with daily and annual cycles, as well as shower activity; that the link was more robust than HF in electrical storms; and that it could be effectively operated using low-gain (5-element Yagi) antennas.

Three other early experimental systems merit attention, the NBS (National Bureau of Standards) system in America described in [12,13], the COMET system which operated between the South of France and the Netherlands, described in [13], and the Hughes Aircraft System described in [14]. NBS system results indicated the advantages to be had with the use of adaptive data rates, while COMET's use of frequency diversity and an FSK scheme allowed for transmission of 150 character messages with average delays of under a minute. The Hughes Aircraft System,

developed under a US Air Force contract, demonstrated the feasibility of using meteor-burst for air-to-ground communications.

The development of operational systems was limited by technological progress however, and the advent of satellite communications led to a serious drain of people and resources. Gilbert [15] describes the situation in a paper entitled 'Growth and Decline of a Scientific Specialty'. In 1957 twenty-four papers on meteor communications research were published in major journals, by 1964 there were just two. In the 1960's most significant work done in the field was by a single team led by GR Sugar at the U.S. National Bureau of Standards [16,17,18].

The data processing and storage limitations of the 1950's and 1960's was overcome by the development of cheap memory and microprocessors during the 1970's. This led to renewed interest in meteor burst as a feasible data communication technique, and to the development of a number of operational systems.

Important recent work has included Abel's study of performance bounds [19], a number of experimental and theoretical studies on characteristics and performance of the channel by Weitzen et al [20,21,22], as well as studies by Ostergaard[23] and by Millstein et al [24] on channel communications potential. Active research into meteor-burst communications is currently being carried out by companies such as Meteor Communications Corporation, SAIC and GE Aerospace in the United States, and Salbu in South Africa, as well as by various defence establishments and academic institutions.

Current applications include *'gathering surveillance, meteorological and environmental data from ground stations and buoys, emergency and rapid deployment communications, primary and back-up communications in the geophysically disturbed auroral and polar cap regions and military communications'*, [25].

1.2 Why Meteor-Burst?

Meteor-burst communication (MBC) has several distinct advantages over more conventional channels such as high-frequency radio or satellite. These advantages can be loosely classified as being of either 'military' or 'non-military' interest, although there is a certain amount of overlap.

1.3 Military Application Interests

The military benefits of MBC have been studied in detail by, amongst others, Hellweg [14], Oetting [26], Whittaker [27], Gray [28], Richmond [29], and Boyle[30].

One factor highlighted by all these studies is the relative immunity of MBC to ground intercept or jamming. Due to the extremely small footprint of MBC signals, (estimated by Whittaker [27] to be about 50km by 20km), a site would have to be exceptionally close to hostile installations for interception to take place. Jamming by hostile forces using a meteor-burst mode would also have little chance of success, as the jamming signal would have to arrive at the same time as the desired signal. This is unlikely to happen unless the hostile transmitter is utilising the same meteor trails as the friendly one, which it cannot do unless it has near-identical system characteristics. Such characteristics would include location, and in this situation, as Hellweg [14] puts it, *'the jammer could be detected and neutralised'*. Gray [28], cites MBC as ideal for communications from forward sites within hostile areas during wartime.

A major advantage of MBC over satellite communications is the lack of vulnerability of the link. It is currently possible for a hostile force to destroy satellites, it is not currently possible for it to destroy some billions of meteor trails.

The fact that remote sites would in general transmit using different meteor trails means that signals to a receiving station would be received at different times. In the rare event of collisions, each site involved would have to wait for the next trail available to it before retransmission. As Johnson [31], points out, *'this is built-in time division multiplex'*.

A last, primarily military, concern is identified by Hellweg [14], Johnson [31], and Gottlieb [32]. Essentially this is that, while most other forms of communications would be rendered inoperable by an atmospheric nuclear detonation, MBC would still be viable, and might even be enhanced, under such circumstances.

Determining the extent of current military use of MBC is unfortunately impossible, due to understandable security concerns. The main supplier of MBC equipment in the United States, Meteor Communications Corporation (MCC) in Seattle, has been most generous in supplying time, advice and information regarding their commercial operation. They also supplied a security officer as a constant escort during a visit, in order to ensure no inadvertent access to the military side of their operations.

What is known is that there is extensive military use of MBC by elements of the US defence establishment, particularly in Alaska, [13], and that studies have been carried out there in the use of MBC as a backup to satellites for the American strategic early warning system [13]. The Chinese military is also known to make extensive use of MBC systems [33], and MBC has been chosen to provide the backup mode for the SHAPE broadcast system in Europe [28].

1.4 Non-military Application Interests

There are several attractive features of MBC for commercial applications, not least of which is cost. As pointed out in [25, 27], the cost for a two-way MBC system would be in the region of £80000, as opposed to the millions involved in setting up a satellite system. This makes it an extremely viable alternative, particularly for third-world countries. Egypt has recently installed a large-scale MBC network of over 200 stations and many other third-world countries are showing interest [34]. Certainly for countries with inadequate telephone and telegraph systems it is cheaper to have remote villages served by MBC stations than it is to lay landlines.

Many current MBC networks are based on having one or more master stations and a number of remote stations. A master station will continuously probe its remotes. The remote only transmits if it receives a probe, indicating that the existence of a meteor trail in the correct region of sky has currently enabled the link to the master, and it has information to send.

An advantage of MBC is the robustness of the equipment needed for such remote stations. A prime example here is the SnoTel (Snowpack Telemetry) Network, which is operated by the U.S. Department of Agriculture and has been described by Barton and Burke [35], Crook [36] and Day [37]. The network is primarily designed to capture information on mountain snowpacks, which provide over 70% of the water supply to areas in the American West. The network comprises two master stations, located at Boise, Idaho and Ogden, Utah, and some 500 remote stations.

In general the sites consist of automated sensors linked to the stations, with no human presence. Many are in mountainous areas and other positions where regular maintenance is not feasible. The remotes are solar powered, and designed to meet a specification that there will be at least one year between service needs. The MBC

systems have been adequate to this task for over a decade, and performance has well exceeded initial expectations. Initial system specifications required a response time for remote stations of less than one hour, the average delay encountered in actual operation has been less than two minutes [13]. The SnoTel network is currently in the process of a significant expansion of operations.

Meteor-burst stations can operate with relatively low transmitter power. As Morgan [38] states, *'the advantages of long range, low peak transmitter power, and equipment simplicity make this technology a candidate for numerous remote and automated sensing stations'*. The low power requirements for remote sites also has a significant bearing on the physical size of systems [39]. With 'boxes' as small as 45x45x30cm, and no necessity for high-gain directional antennas, MBC remotes can, and have been, successfully mounted on such platforms as light aeroplanes and ground vehicles [14].

As stated in [25], meteor-burst links can operate effectively using only a single frequency, regardless of sunspot activity or the period of day or year. This contrasts strongly with HF radio, where frequency must be constantly altered for optimum range and minimum losses.

A final advantage of MBC is its robustness in the face of atmospheric conditions which seriously affect other communications, such as HF radio. In particular, auroral and polar disturbances have little effect on MBC [10, 14], which accounts for its high level of deployment in areas such as Alaska. The Alaska Meteor-Burst Communications System, (AMBCS), which is jointly operated by five different U.S. government departments, and described in [13, 14], is a prime example.

1.5 Disadvantages

No MBC transmission can take place unless a meteor trail is in the correct area of sky between stations. The average time between trails will vary according to yearly and daily cyclical variations in meteor arrival rates as well as features such as galactic noise, transmitter power, and antennas employed. Current systems have average delays between usable trails varying from less than a second to as much as a minute. These delays are a negative feature for applications where time-critical messages are sent. Despite that, it is worth noting that in 1984 US Air Force tests, MBC links averaged six seconds behind satellite links for transmission of aircraft tracking information on a Tin City to Anchorage Alaskan route [13].

While meteor trails can support high data rates - up to megabits per second in some cases [20] - the generally short duration of a couple of hundred milliseconds means that the effective data throughput of current systems is at teletype levels. Great improvements in the data capacity of the channel have been made in recent years, as will be discussed in this thesis. However, despite some ingenious attempts, (see [40]), MBC systems do not currently allow for voice transmission, or indeed for rates much higher than 200 baud. This is clearly inferior to HF or satellite alternatives for applications where higher data rates, voice or video transmission is required.

2 The Classification of Meteor Trail Reflections

Abstract

This chapter describes a detailed new classification schema for meteor trail reflections. The schema allows fine distinction among identifiable subtypes within previously discovered families, as well as the determination of previously 'unknown' types. Previous work done in the field is summarised with the importance of such a schema being highlighted. Some statistics showing the relevance of the classifications defined are presented and discussed. The design and implementation of the rule-based expert system used to obtain the classifications, together with its applicability and advantages in this environment, is discussed in some depth.

Material in this chapter has previously been published in the paper 'The Classification of Meteor Trails by a Rule-Based System' by SW Melville, JD Larsen, RY Letschert and WD Goddard, in Transactions of the SAIEE, Vol 80, No 1, September 1989.

2.1 Introduction

It has long been established that there are different families of meteor trail reflections [18] which can be classified according to features such as shape, duration and amplitude (see [9], [41]).

Ostergaard [23] defined five categories, or types, of meteor trail reflections - underdense, overdense, 'tiny', sporadic-E and a catch-all group, 'other'. He states that '*some of these classes contain waveforms which agree closely with the classical theory of meteor scattering ... other classes ... cannot be associated unambiguously with a separate physical mechanism of propagation. However, they occur often enough to warrant separate classification.*'

Such a classification system is described in this chapter. In addition to being able to identify several families within Ostergaard's 'other' group, the system recognises a number of sub-families within the previously known groups.

Type determination is of major importance to many aspects of meteor scatter study. In 1986, Weitzen and Tolman [41] described an automatic classifier which classified trails forming the basis of further study [22]. They state that "*The classification procedure ... is an important function since the different propagation mechanisms and different types of meteor trails have different communication properties.*"

While this is certainly valid it is felt that their classification (primarily into underdense, overdense, and non-meteoric groups) is too coarse to allow an absolutely reliable study of meteor scatter systems. Meteor scatter system analysis based on measured data is normally presented in the form of counts of meteors and the distribution of characteristics such as duration, amplitude, time constants, and wait time of these meteors. It is clear that results on a particular group of meteors (eg. underdense) can be seriously compromised if 'imposters' are included in the group. (Fragmented overdense trails could have serious effects on an underdense wait time distribution, for example.) At the same time, ignoring all trails which do not comply exactly with a set norm is just as unsatisfactory. (A recognisably underdense trail which has some feature, such as multiple plateaus, distinguishing it from the classic underdense model, still needs to be considered in statistics on underdense trails.)

In addition the distribution of time constants on the falling slope of underdense trails can be seriously affected through sharp fades arising out of wind distortion or through irregularities in the latter part of the trail. These irregularities would normally be missed in a coarse classification schema.

In analysis carried out on underdense trails three regions are normally considered - the rise, the peak, and the fall - and since these three can be distorted through a number of physical mechanisms, analysis should take into consideration the various trail types which exhibit irregularities in one or more of these regions. A significant problem in analysis is that these distortions are normally included in statistics of measured trail data over all trails. A detailed classification schema allows the identification of these trail types and thus gives the researcher the ability to selectively remove their influence from statistics (for example, trails with distorted fall regions could be removed from those considered in analysing fall regions) as well as the chance to develop statistics exclusively on these types.

Identification of modes of propagation that are not meteoric is imperative since these could quite seriously affect statistics. However in certain cases these modes of propagation can closely resemble meteor scatter reflection and therefore a close classification schema is required in order to identify such reflections. A second importance of the identification of other modes of propagation is that these do not exhibit the normal footprinting effect found with meteor reflection and therefore could seriously reduce the intercept immunity of meteor scatter communication. Therefore in a case where such effects are being investigated a fine classification schema would be most useful. As pointed out by Ince [42] an improvement in meteor scatter systems performance is possible through employing antenna space diversity. This improvement is primarily a result of the reduction of multi-path effects. These effects typically result in 'fast fading' in signal amplitude. The determination of the percentage of trail reflections which exhibit this characteristic will allow an evaluation of the importance of this factor.

In the course of this chapter the method of determining families is described, and a brief overview of the determinable families is given; finally statistics concerning the relative importance of these families are presented. Some speculation as to how the

various families arise is indulged in, however the main work here involves the identification of types rather than the determination of their origins. Note particularly that the names used to describe the various families are based primarily on shape description, rather than origin description. Thus a name such as 'square-root sign', while giving no clue as to reflection origin, (which would at best be speculative), does adequately describe the 'shape' of the reflection when displayed on time and amplitude axes. This approach was taken as it soon became obvious that description by propagation mechanism would be tentative at best for some types, while to describe the origin of 'distortions' in some sub-families (plateaus, 'humps', etc) would also have involved a degree of guesswork.

The design, implementation and results of 'TrailStar', a flexible rule-based system is discussed in this chapter. There are three major aspects to this system:

- The system allows automatic classification of meteor trail reflections according to a specified schema.
- The users of the system are at liberty to alter/improve this classification schema, and a number of aids for this purpose are embedded in this system.
- The current schema employed gives a consistent and fine classification of meteor trail reflections

2.2 Data Source

The data used as a basis for the work done on trail reflection classification was gathered on a monitoring system developed by the Electronic Engineering Department of the University of Natal. The measurement system was designed to allow the development of detailed statistics on the performance of meteor scatter systems in the southern hemisphere [43].

Relevant aspects of the monitoring system are as follows :

- Transmit power 400 W
- System noise figure 2 dB
- System bandwidth 2 kHz
- Data sampling interval 5 ms
- Maximum signal strength -80 dBm
- Frequency 50 MHz

The basic computer monitoring equipment consists of an analog-to-digital converter, processor unit, memory, printer and display. The system has been designed in order to measure detected signal strength in dBm at five millisecond intervals.

The monitoring system will trigger on any trail reflection provided it remains above a required signal-to-noise ratio for a specific interval. The threshold and duration above threshold are set by an operator and the values used for these, in the data analysed in this work, were 10 dB and 20 ms. Once the system has triggered on a trail reflection it will log the signal strength at five millisecond intervals from the point the reflection rose above the threshold to the point it falls below a second threshold for a required interval. The end of reflection threshold and duration below threshold may also be configured by the operator. In the gathering of the data used it was found that an end of reflection threshold of approximately 9 dB and duration below threshold of 400 ms was required in order to prevent the system from fragmenting overdense trail reflections. The system is therefore able to log trail reflections from the smallest underdense trail reflections which have durations above threshold in the range of tens of milliseconds to the longest overdense trail reflections which can endure for tens of seconds.

Since the majority of the data was recorded in a quiet rural noise environment the noise power (2.0 kHz bandwidth) was typically in the range -130 dBm through to -120

dBm which would in turn correspond to a minimum detectable trail reflection strength in the range of -120 dBm through to -110 dBm.

Sampled trail reflections are termed 'reflection envelopes' and are stored to disk for future processing. In addition to the reflection envelope, fundamental information about the trail such as time of occurrence, peak amplitude, duration and noise floor are also recorded in an associated header.

Data were gathered over both 550 km and 1100 km links, and should thus be representative of what would be found in typical meteor scatter systems. Over the 1100 km path up to 10 000 trails were recorded per day while over the 550 km path the maximum number of trails per day was approximately 5000. The total number of trails stored to disk using the monitoring system is in the order of several million at the time of writing this. The table below specifies the transmit and receive sites and antennas used for the particular data presented here. (Note that the antennae specified below are horizontally polarised.)

Link	1100km Midpath	550km Midpath
Tx Site	Pretoria (26°S, 28°E)	Pretoria (26°S, 28°E)
Rx Site	Arniston (34°S, 20°E)	Durban (30°S, 31°E)
Tx Antenna	Stacked 5-element Yagis (9 dBi each)	11-element Yagi (12 dBi)
Tx Antenna Height	Lower 9m, upper 14m	3 mRx Antenna
Rx Antenna Height	Lower 9m, upper 14m	3 m
Rx Elevation	0°	0°

Table 1 - Measurement Links

2.3 Overview of the Typing System

The TrailStar system was developed to allow a flexible and efficient means of typing meteor trail reflections. This rule-based expert system can be run on an IBM PC or a compatible, thus allowing immediate classification of captured data in the field. The system was written in TurboPascal (version 4.0).

The system will be discussed in greater depth later, for the moment it is just worth mentioning that the system is so designed to allow fine-tuning by an expert human classifier, and thus has been designed with the issues of learning, ('*knowledge acquisition lies at the heart of the design and construction of expert systems*' [44]), and user interface optimisation being of major importance. This latter aspect is often overlooked by system designers, however it is of vital importance for any system with significant user interaction. Buchanan [45] perhaps puts this best when he states, '*Human engineering issues are important for making the program understandable, for keeping experts interested, for making users feel comfortable. Explanation, help facilities, and simple English dialogue thus become important.*'

2.4 Approach

The first thing that was done was to get a pictorial display of the meteor trail reflections available to the human classifier. (An explanation of the display used as well as some typical trails appear later in this chapter.)

Once this was done, it seemed that the most effective approach would be as follows:

(1) Classify trail reflections automatically according to some set of criteria

(2) Check classifications assigned manually to see if domain expert in agreement

(3) Repeat from (1) until satisfied with criteria used

Initially it was believed that only five or so distinct types of meteor trail reflections would be encountered, and the intention was to determine trail type primarily by an underdense/overdense distinction. However it soon became apparent, after viewing trail reflections pictorially, that there would be many more identifiable types of trail reflections involved. A flexible approach was clearly indicated, and this expanded the task to become not only one of finding and implementing a one-off classification system, but allowing for narrower classifications and new type definitions by any expert user prepared to spend time 'fine-tuning' the system. Importantly, it was felt that such an expert could not be assumed to have any great computer expertise, and that the system could be used and refined without needing alterations to software.

A rule-based expert system approach was clearly indicated here, the design and use of which will now be discussed.

2.5 The TrailStar Rule-based Expert System Design

This system is based primarily on rules. A rule consists of a set of conditions and a single action. The action is a simple assignment of a particular classification to a meteor trail reflection, while conditions return a true or false result depending on whether or not the aspect/feature they test for is satisfied in a particular trail reflection.

Rules are tested according to a priority order, with the first rule 'triggered' (having all its conditions being true) being the one 'fired' (having its action applied).

It is important to note that the set of rules ('rule base') is not static but can easily be modified and enlarged as needed. Indeed any user can, without altering computer code, implement such changes.

The system can be seen as consisting of three conceptual levels. At the 'bottom' level is a set of routines which generate numeric values describing various features of a trail reflection (for example, the position of the peak relative to the overall length of the trail reflection). At the 'middle' level are the conditions, which are routines which test these numeric values against given values/parameters, returning a true/false result. For example, a condition could test whether the peak lies in the first fifteen percent of the trail reflection. Finally, at the 'top' level are the rules themselves.

To define a type the user creates a rule or rules by listing the conditions (feature descriptors) which are sufficient, if true, to allow type determination, and places the appropriate action in the rule. For example, a currently used rule is:

CONDITIONS :

[1] Trail duration less than or equal to (40) times 50 ms.

[2] Peak is in (1st) third of trail.

[3] Straight line variance over trail (3) times variance over fall.

[4] Straight line variance over trail (2) times variance over rise.

[5] Amplitude range over trail is greater than (3) dBm.

[6] Straight line variance is greater than (8) times 0.1. dBm squared.

[7] Upper plateau is not present.

ACTION : Trail typed as CLASSIC UNDERDENSE.

Many of the conditions are parametrised, (parameters in the conditions above shown in parenthesis), so the user can adjust parameters to alter tolerances within a rule.

After creation of a new rule the user must then determine what its priority in the rule-base should be. This priority ordering approach allows a fall-through situation, (for example the rule giving the 'trail type unknown' action would always have lowest priority), and also allows the thinking user to order his/her rules in such a way as to

ensure minimal processing time. (As a general practice the most common types should have their rules appear first, while rules with conditions requiring serious computation such as multiple parabola fits should only be tested after all others have failed to trigger.)

The user can of course add conditions to, or delete conditions from, rules at will, as well as adding and deleting the rules themselves.

2.6 Justification and Explanation of Approach

The rule-based expert system approach is particularly well-suited to the problem for a number of reasons. First and foremost, the trail reflection types were neither well-defined nor necessarily invariant. In other words, on beginning the problem there was no idea as to how many different families would be found, and when defining a family in a broad sense there was uncertainty as to whether there would be sub-families within it. (The case of 'underdense trails' was an extreme example of this, where more than ten distinct sub-families arose out of a single original family.)

An advantage to such an approach is that an user can alter rules and priorities in order to iterate to an acceptable classification. As Feigenbaum [46] puts it, *'the rule-based approach allows for great flexibility for adding, removing, or changing knowledge in the system*'.

With such an approach the classification becomes a task of defining rules, and scanning the groups ('buckets') of the various types of meteor trail reflections determined by the rules. (While trail reflection records are not actually physically grouped according to type, the classification process alters a field in the trail reflection record to indicate the type involved. The software can then easily allow for 'scanning' by only showing those records with type field corresponding to that under consider-

ation.) Where there seems to be more than one distinct type in a 'bucket', a new rule or rules could be determined which would break the type into two sub-types. In addition, the 'unknowns' bucket can be scanned to see whether any group of trail reflections within it has some common feature, which would allow the creation of a new type. Rule creation, typing and scanning as described above continues until the stage is reached where, firstly, there are no obvious patterns in the 'unknowns', and secondly where no strong differences exist within families.

The original rules of the TrailStar system catered for only four basic types - 'classic underdense', 'overdense', 'short mid-peak' and 'gothic rockers' (the bucket containing trail reflections not catered for by the current types). After several iterations a total of twenty-eight different types have been formulated. The stage has now been reached where no obvious patterns can be detected in the 'unknowns' and no strong differences are found within families.

Clearly this is fairly subjective, 'strong differences' and 'obvious patterns' are only strong or obvious by agreement within the research team - other researchers might choose to create even more sub-families or succeed in finding a family or families within the 'unknowns'. However, this is exactly what the rule-based expert system is designed for; by far the largest part of its structure is concerned with catering for new and revised classifications (the acquisition of further knowledge about the domain).

Importantly the classification system has capacities for self-justification and experimentation. That is, if one wants to know why the system decided on a particular classification, one can just ask it, and it will return an English language text explanation of the conditions and action of the rule that was fired to give the classification. (Conditions are stored as numbers within the rule structures for store and efficiency reasons, but text stubs associated with conditions are kept on file to allow clear explanation of the conditions to the user.) As far as experimentation is concerned,

one can also request the 'next choice' classification - the classification that would have been made if the type chosen had not existed. This sort of information is highly valuable for the expert trying to set up his/her rules in such a way as to ensure an acceptable (from the expert's point of view) typing.

2.7 Condition Routines (Trail Reflection Descriptors)

The condition base currently has eighty-three conditions, each of which tests some feature of a trail reflection. The conditions range from simple tests like whether trail reflection duration is greater or less than some user-defined parameter, to complex tests such as whether a certain percentage of parabolae between local minima have a variance below a certain amount. The problem of finding ways in which to describe features of trail reflections was a fairly major one, complicated by the fact that as finer and finer distinctions between families were desired so more differentiating features had to be discovered.

The majority of the descriptors which did turn out to be useful involved conditions in some way related to the following trail reflection features:

- Duration.
- Position and amplitude of the peak signal
- Presence or absence of plateaus.
- Amplitude range.
- Time since last trail reflection encountered.
- Straight line variance over the whole trail as well as over the rise and fall sections of the trail.
- The relative differences between rise, fall and whole trail straight line variances.
- Slopes of straight line fits over rise and fall areas, as well as over the entire trail reflection.
- Position of absolute minimum signal in trail reflection

- Number of fades in the trail reflection.
- The number of low variance straight-line fits needed to cover the entire trail reflection. The variance between the best parabola fit to the trail reflection and the actual trail reflection.
- The number of local extrema.
- The variance of parabola fits between local minima and between local maxima.
- The duration of falls and plateaus with respect to each other and with respect to the whole trail reflection.

Most classifications come about from a combination of tests on the features above, although the current condition base has other conditions available to the user, most often to allow for 'exception' type rules (for example there is an 'unreasonable data' condition which tests if any samples had values outside the range of the measuring system). Also conditions exist which are not used in the current set of rules, but are there so as to be available to users who wishes to expand or alter types.

2.8 Results and Uses

Typical examples of the more important of the twenty-eight trail types discovered are shown in Figures 1 through 18 on the following pages. The display chosen gives both a 'real' and a 'stretched' view of the reflection, in both cases the X-axis shows time in milliseconds and the Y-axis signal strength in dBm. The 'real' view plots the samples described earlier point for point with fixed time increment on the X-axis for all trail reflections, while the 'stretched' view shows the overall shape of the trail reflection with greater clarity over the whole screen. (Interpolation, either linearly or by cubic splines depending on sample points available, along with compaction of exceptionally long trail reflections, is employed here.)

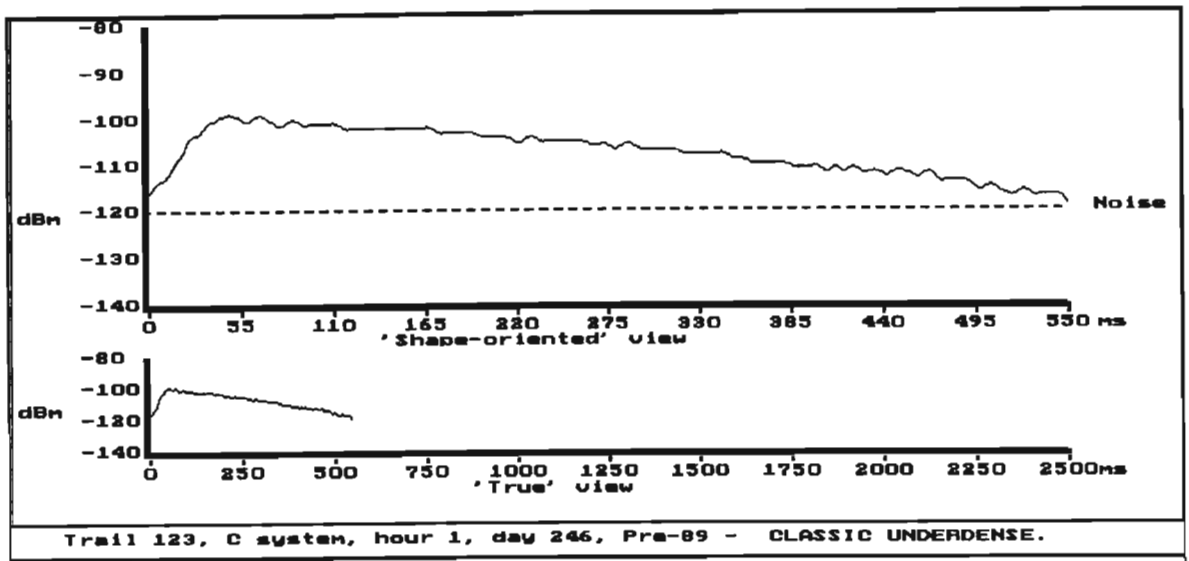


Figure 1 - Trail Type 9 : 'Classic Underdense'

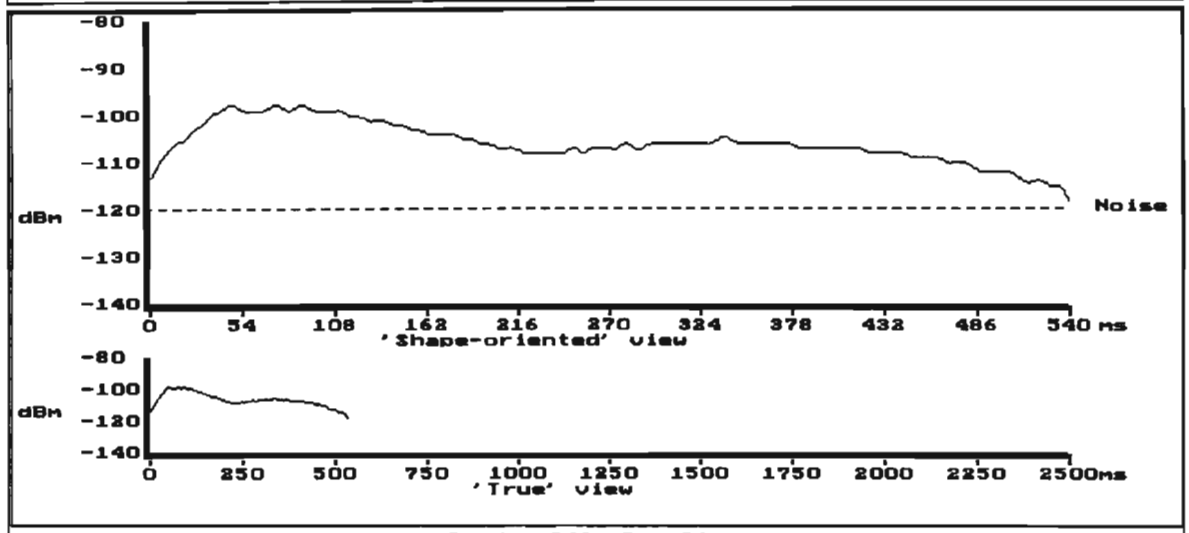


Figure 2 - Trail Type 29 : 'Hump-Backed Classic U/D'

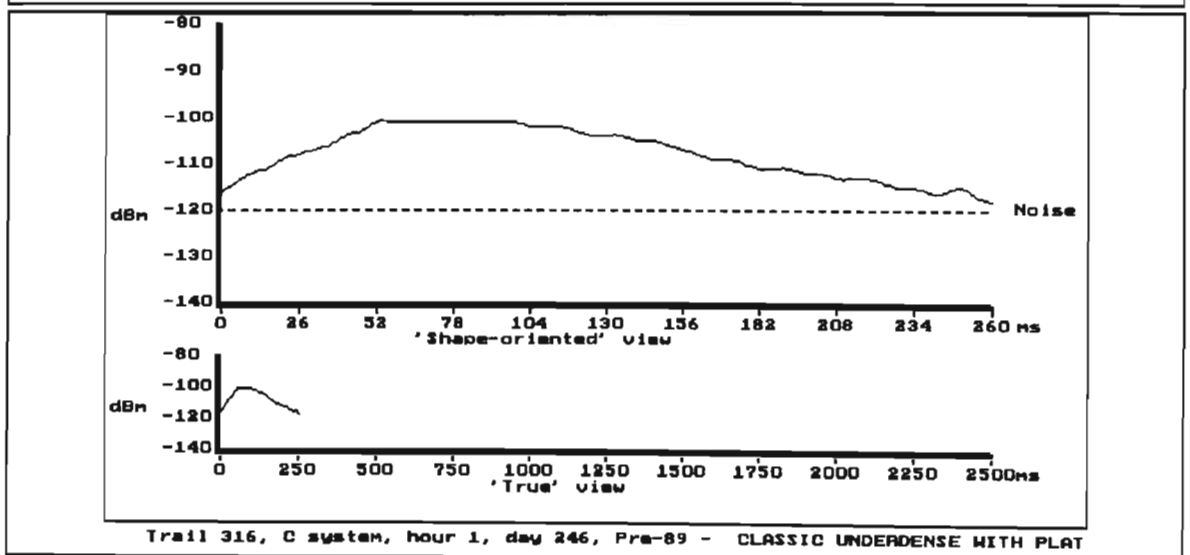


Figure 3 - Trail Type 10 : 'Classic U/D with Plateau'

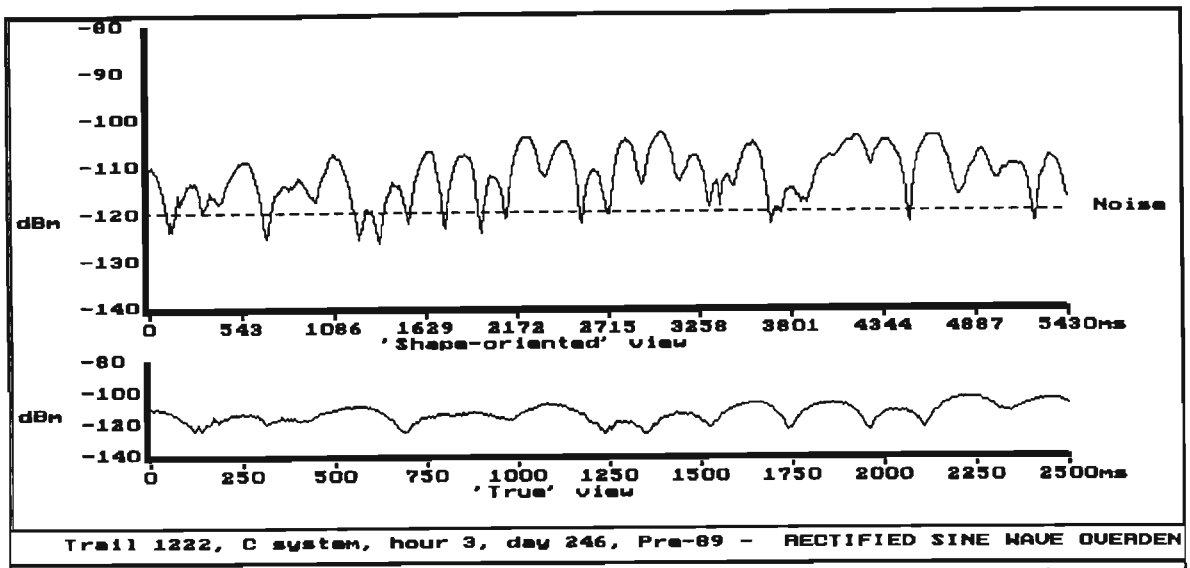


Figure 4 - Trail Type 20 : 'Rectified Sine Overdense'

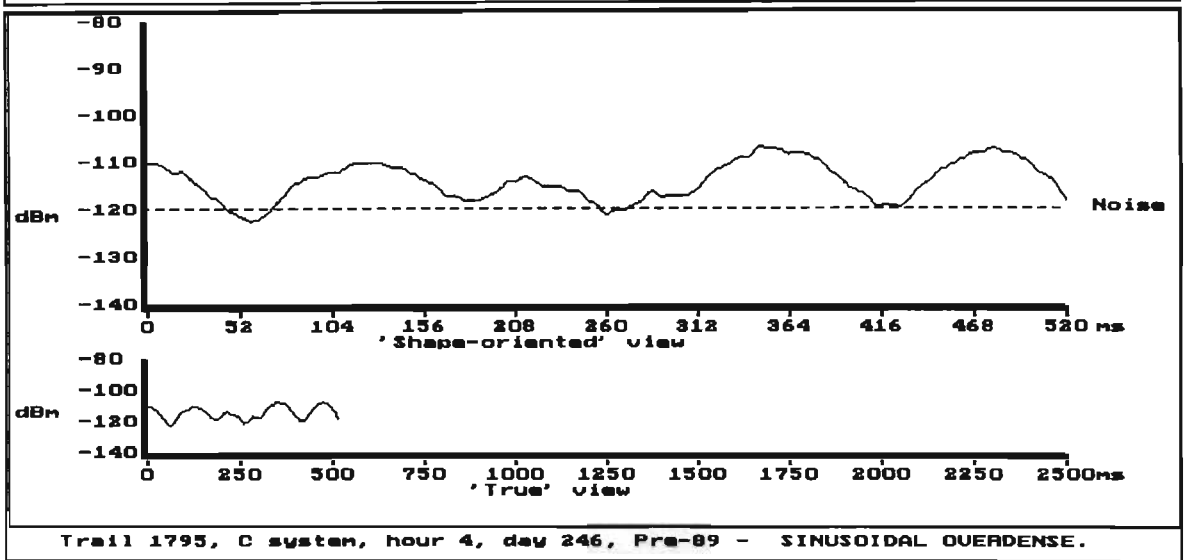


Figure 5 - Trail Type 27 : 'Sinusoidal Overdense'

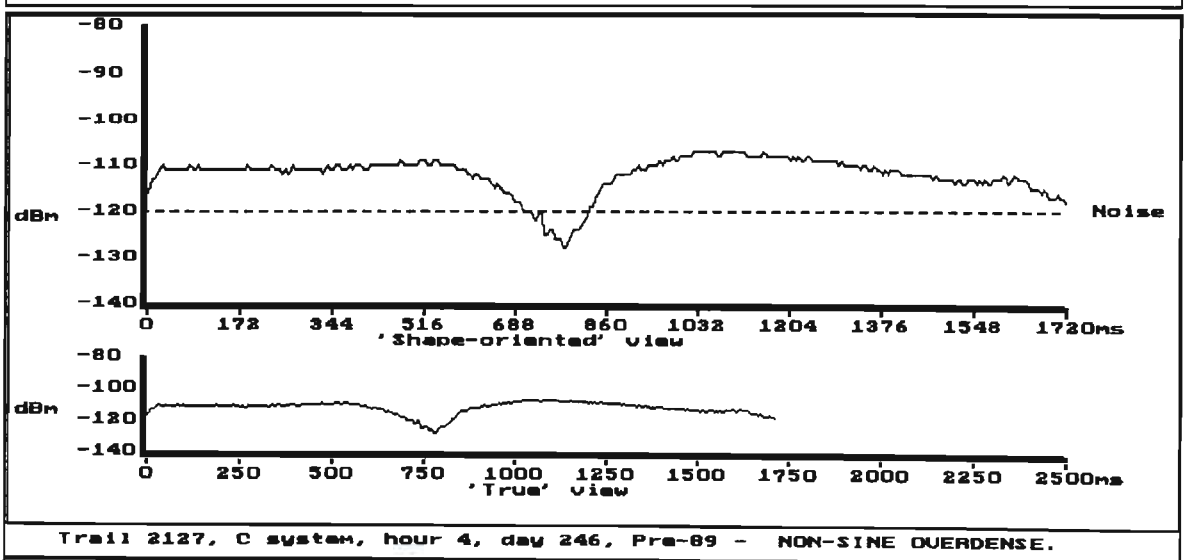
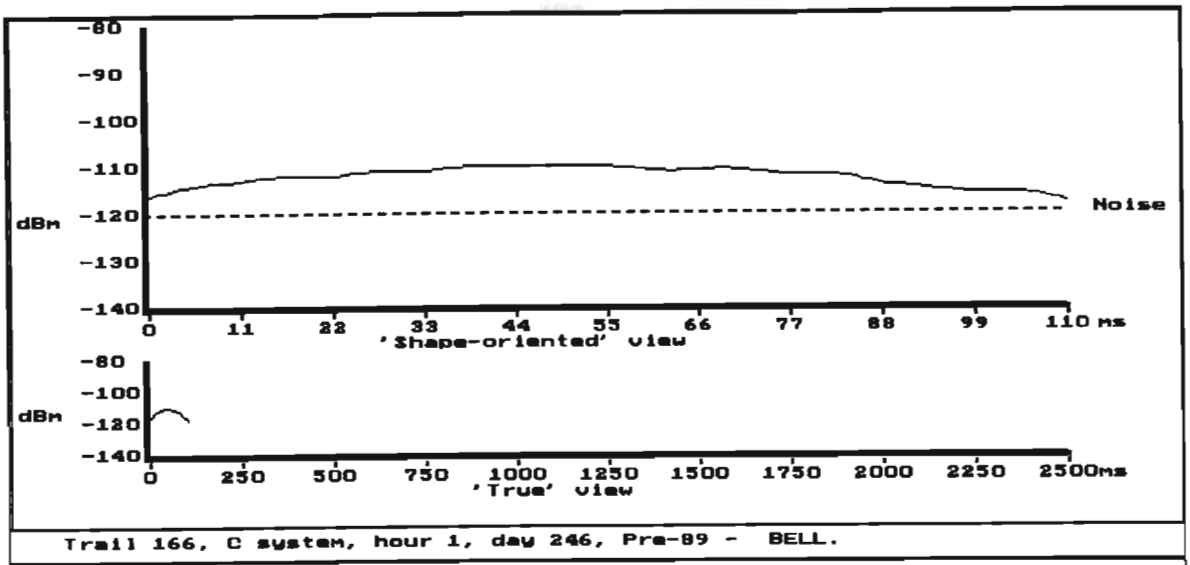
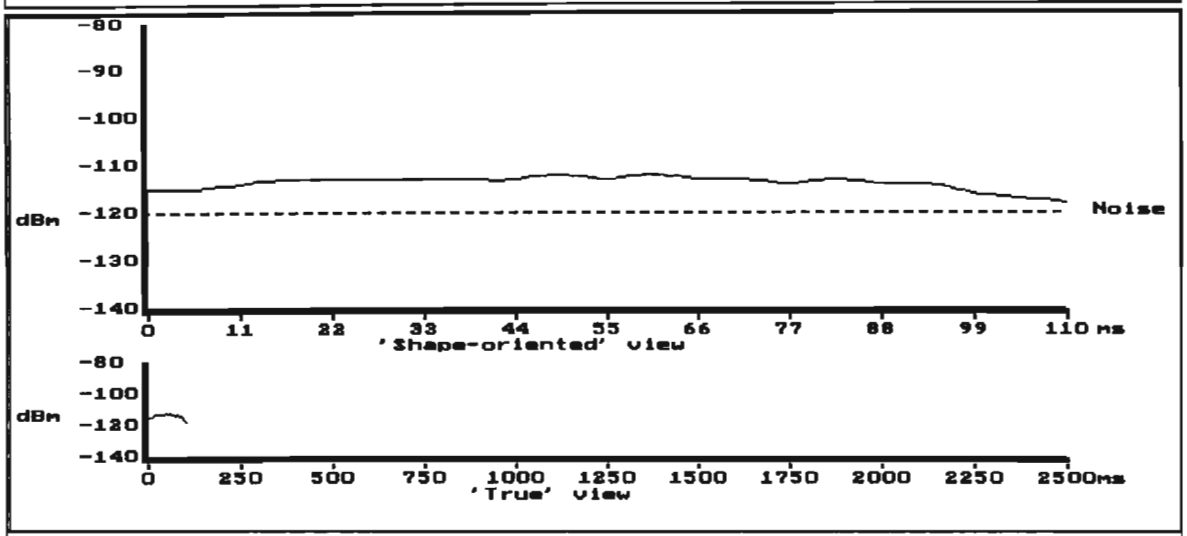


Figure 6 - Trail Type 21 : 'Non-Sine Overdense'



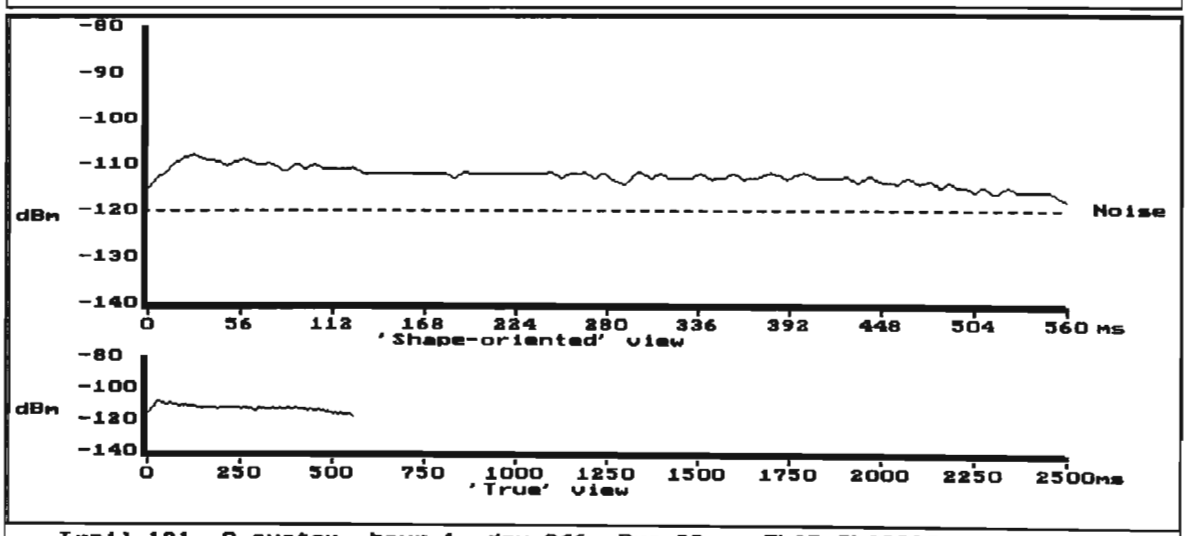
Trail 166, C system, hour 1, day 246, Pre-89 - BELL.

Figure 7 - Trail Type 15 : 'Bell'



Trail 439, C system, hour 1, day 246, Pre-89 - FLAT BELL.

Figure 8 - Trail Type 6 : 'Flat Bell'



Trail 191, C system, hour 1, day 246, Pre-89 - FLAT CLASSIC

Figure 9 - Trail Type 5 : 'Flat Classic Underdense'

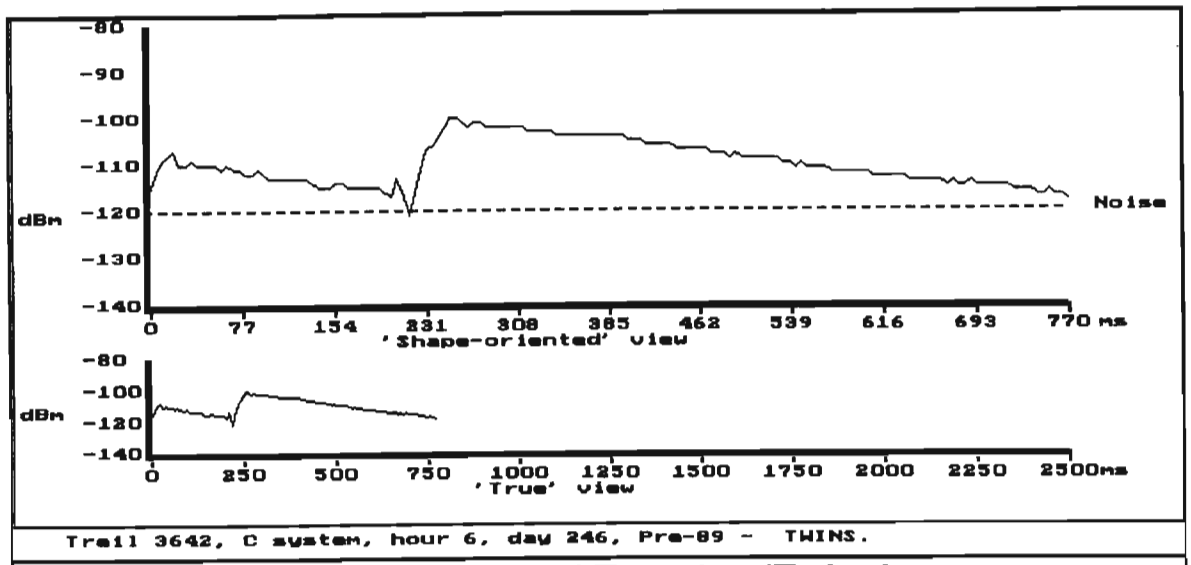


Figure 10 - Trail Type 18 : 'Twins'

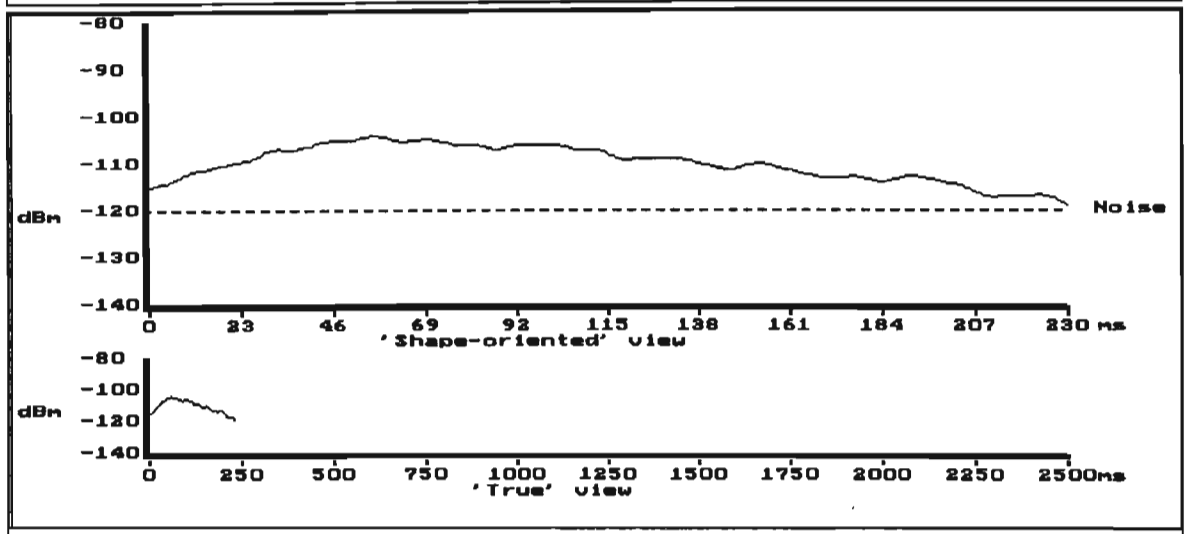


Figure 11 - Trail Type 11 : 'Round-top Classic U/D'

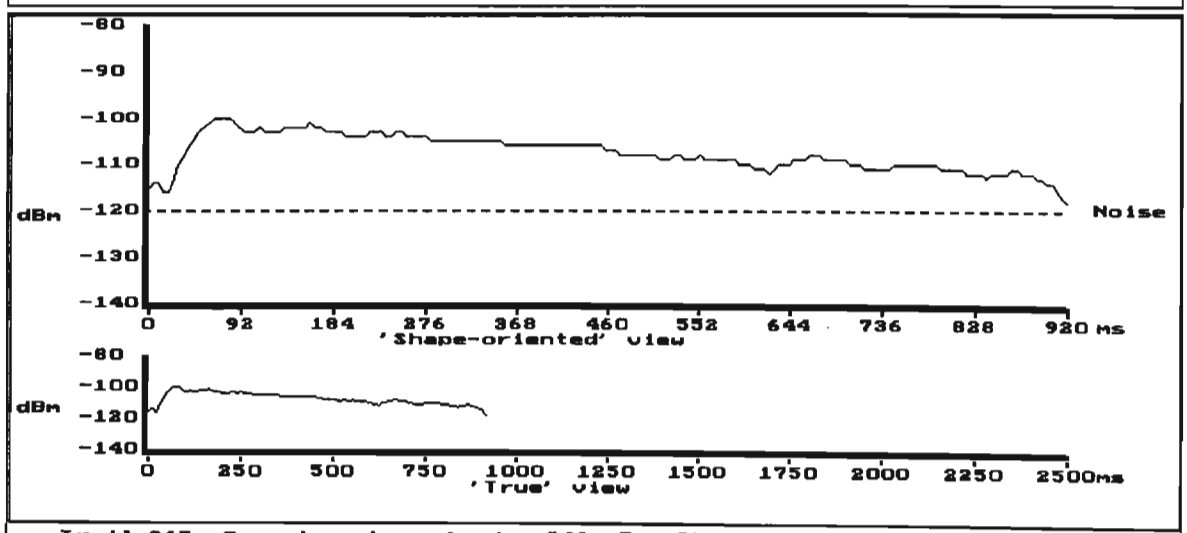
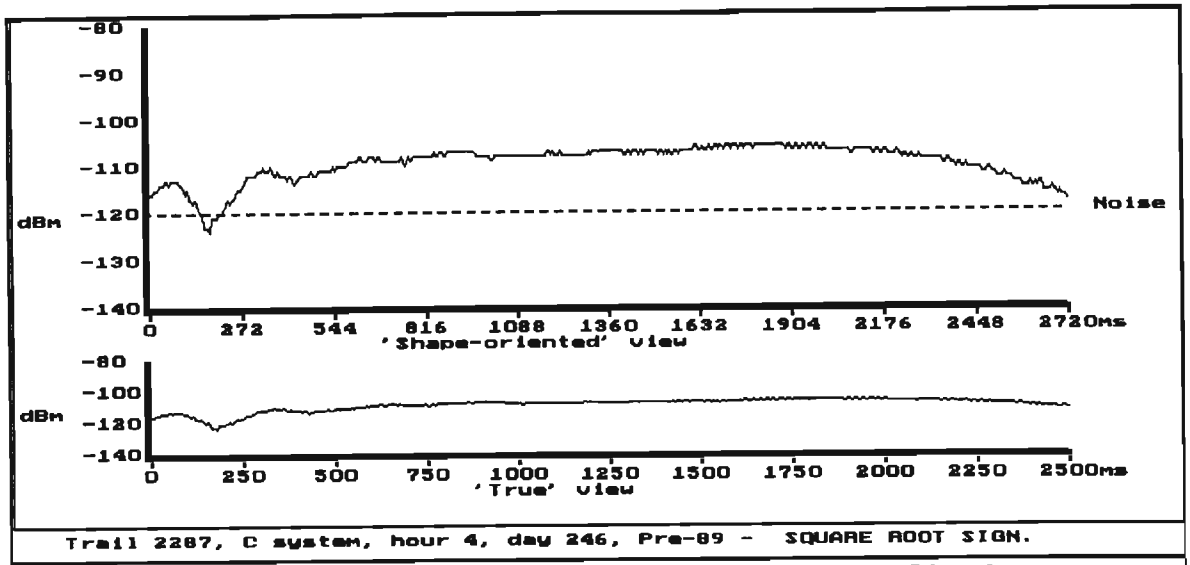
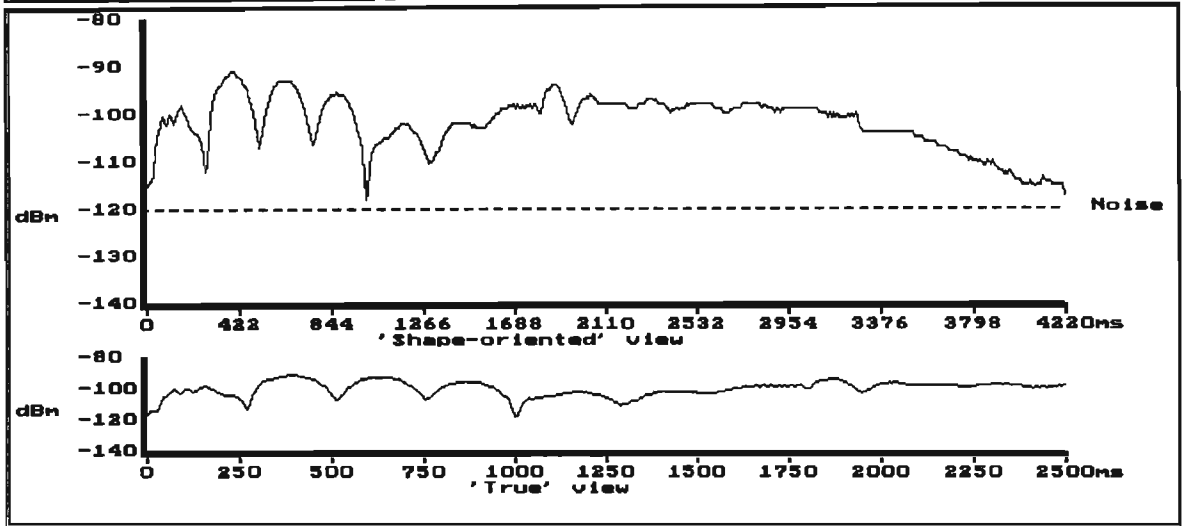


Figure 12 - Trail Type 13 : 'Underdense with Notched Rise'



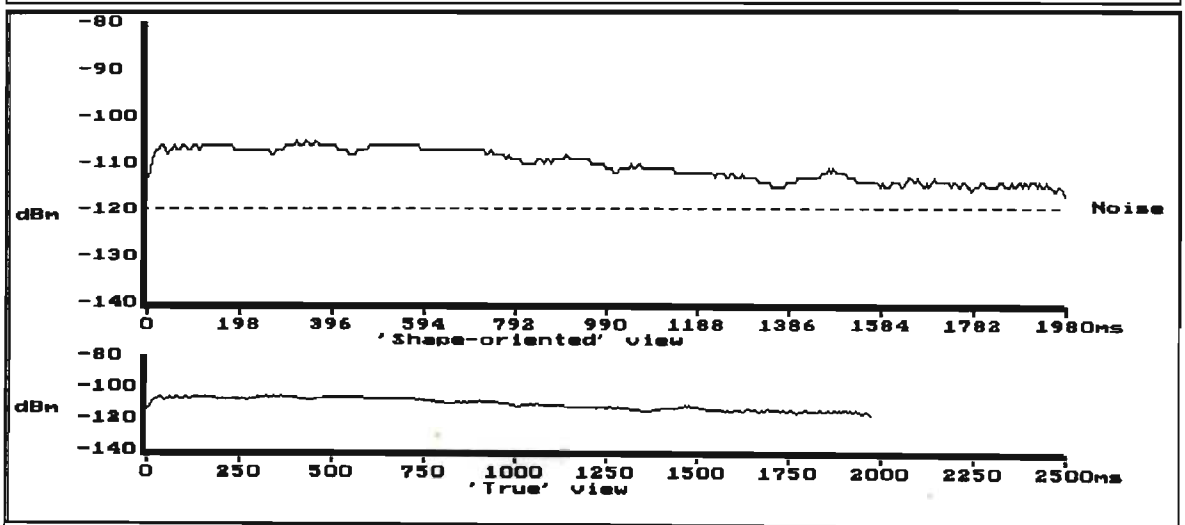
Trail 2287, C system, hour 4, day 246, Pre-89 - SQUARE ROOT SIGN.

Figure 13 - Trail Type 19 : 'Square Root Sign'



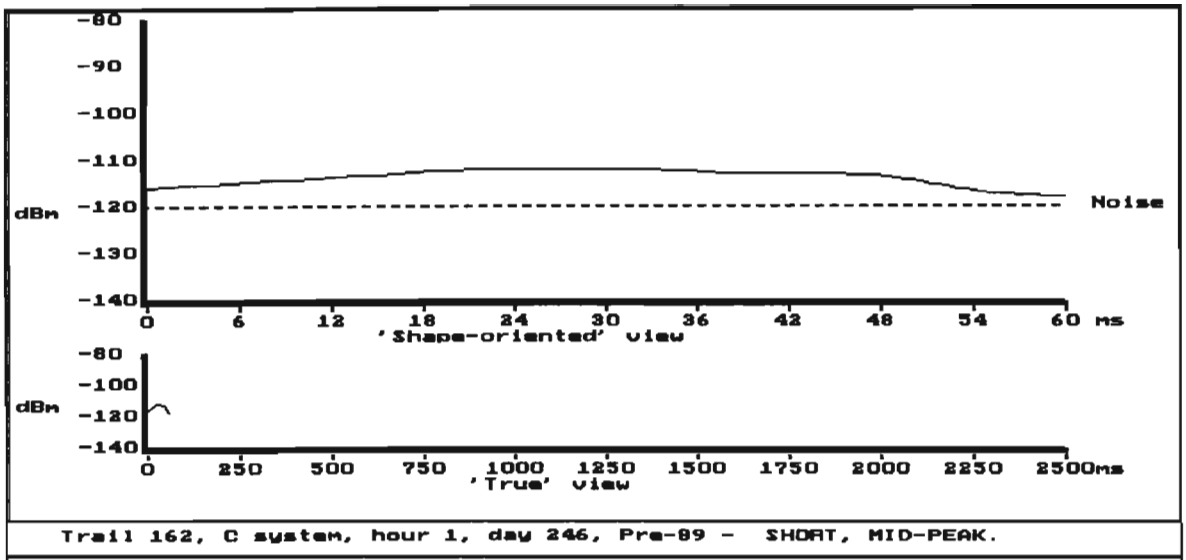
Trail 2848, C system, hour 5, day 246, Pre-89 - WIND-BLOWN OVERDENSE.

Figure 14 - Trail Type 28 : 'Wind-Blown Overdense'



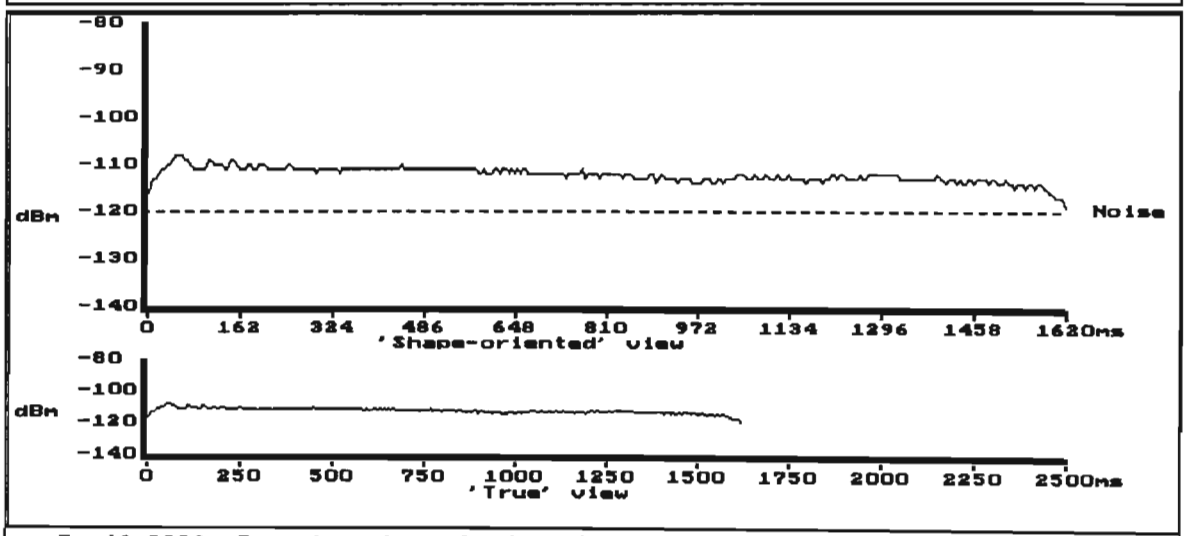
Trail 1072, C system, hour 2, day 246, Pre-89 - HAZY CLASSIC.

Figure 15 - Trail Type 24 : 'Hazy Classic'



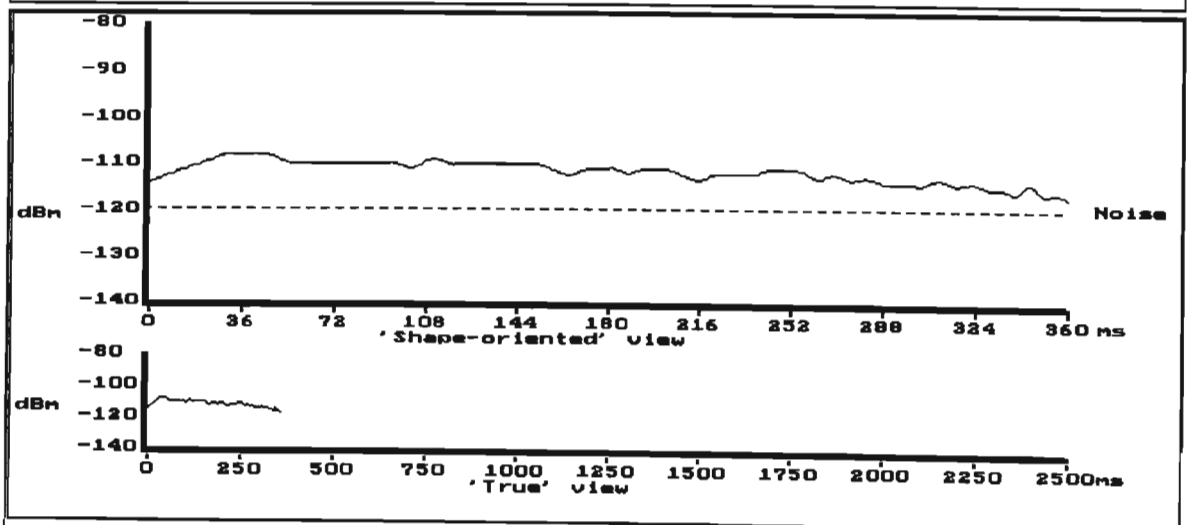
Trail 162, C system, hour 1, day 246, Pre-89 - SHORT, MID-PEAK.

Figure 16 - Trail Type 2 : 'Short, Mid-Peak'



Trail 2986, C system, hour 6, day 246, Pre-89 - MULTI-PLATEAU UNDERDENSE.

Figure 17 - Trail Type 16 : 'Multi-Plateau Underdense'



Trail 1261, C system, hour 3, day 246, Pre-89 - MULTI-SLOPE UNDERDENSE.

Figure 18 - Trail Type 17 : 'Multi-Slope Underdense'

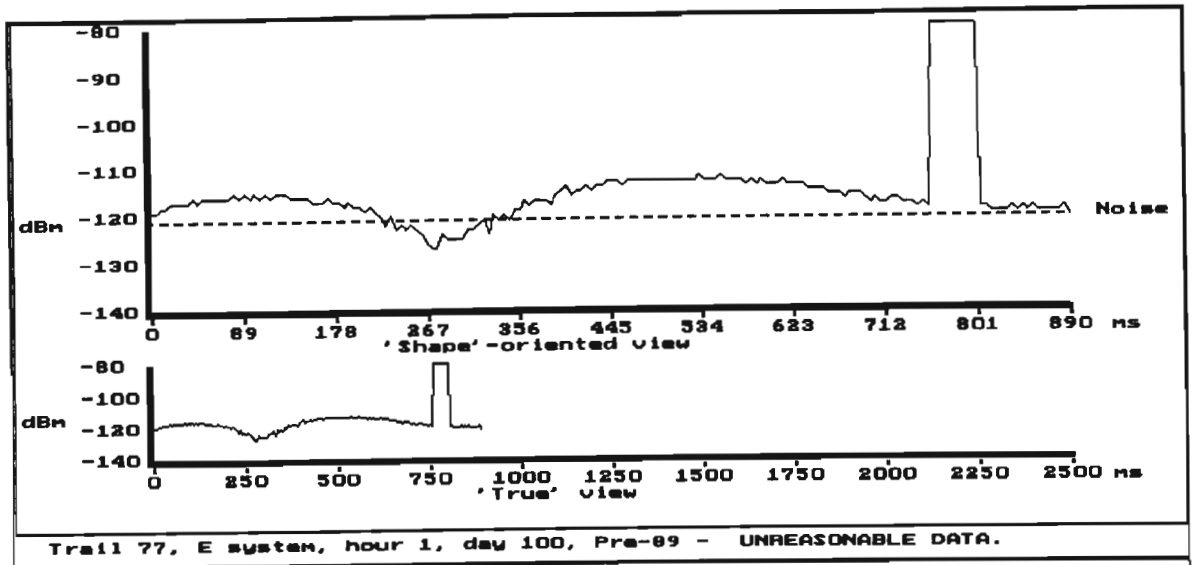


Figure 19 - Trail Type 1 : 'Unreasonable Data'

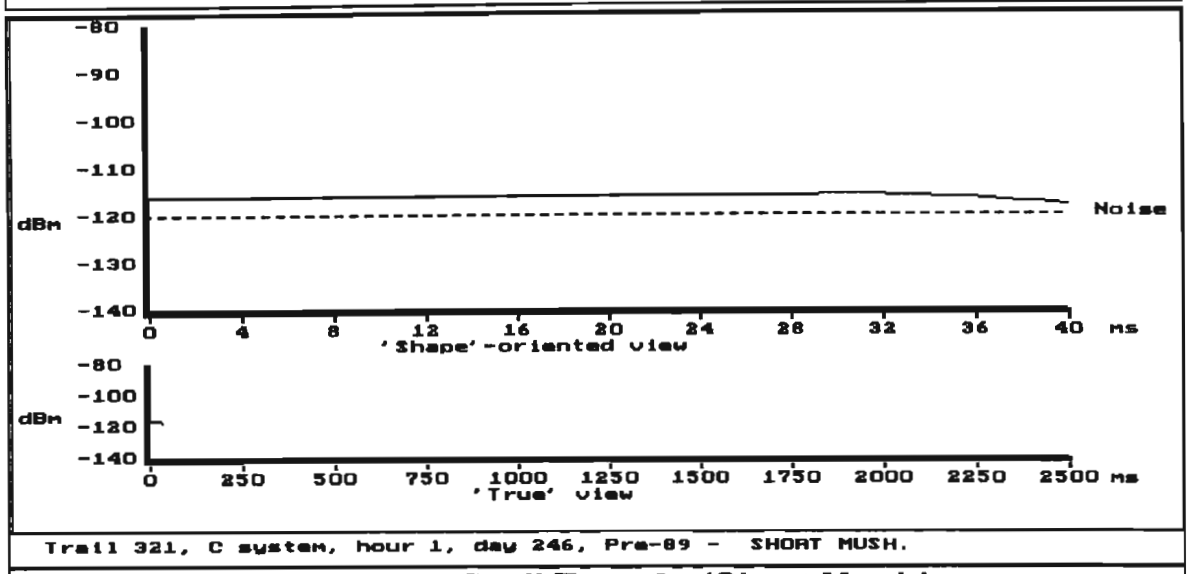


Figure 20 - Trail Type 3 : 'Short Mush'

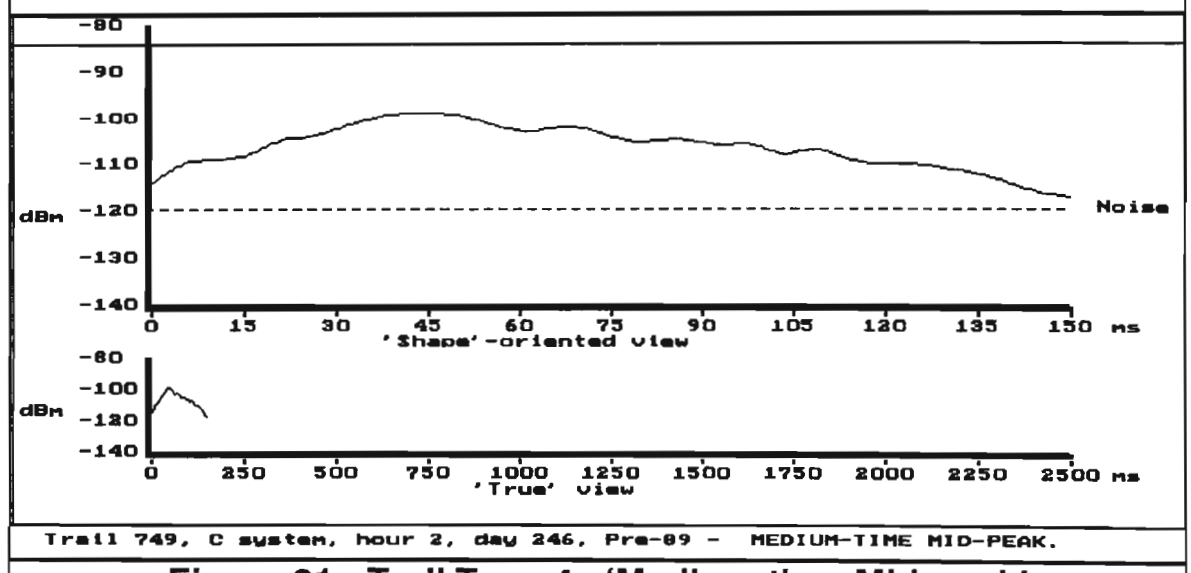


Figure 21 - Trail Type 4 : 'Medium-time Mid-peak'

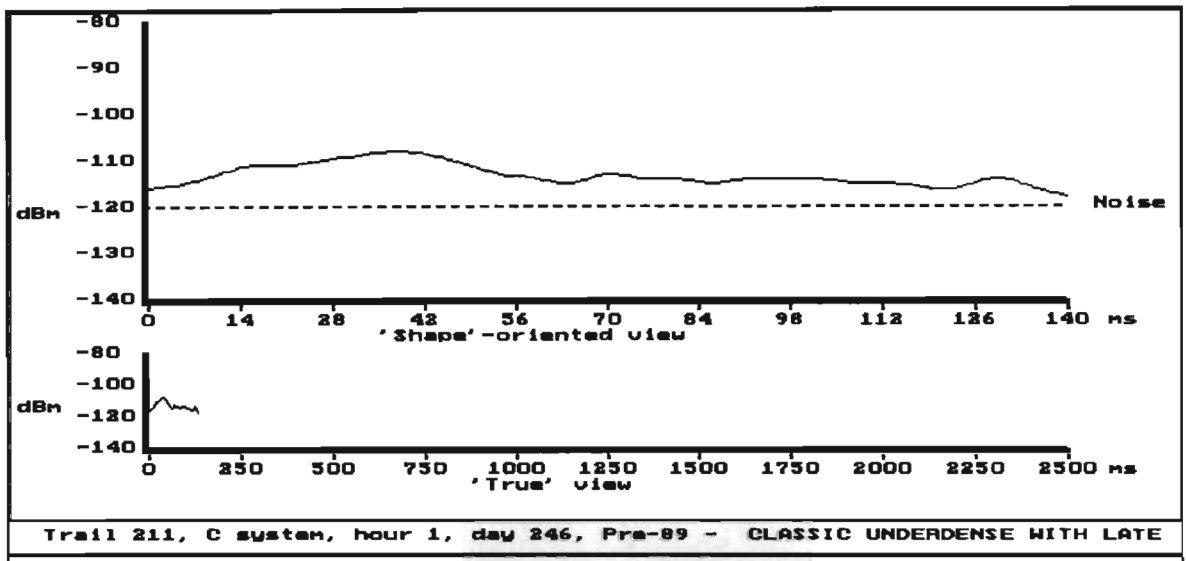


Figure 22 - Trail Type 12 : 'Classic U/D, Late Fall'

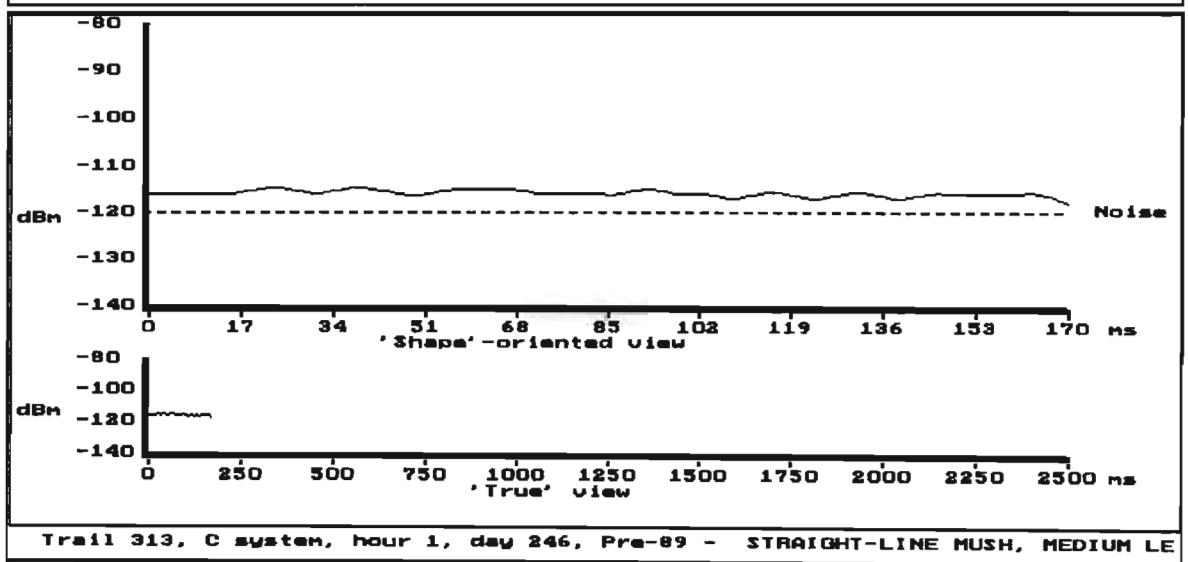


Figure 23 - Trail Type 7 : 'Straight-line Mush (Medium)'

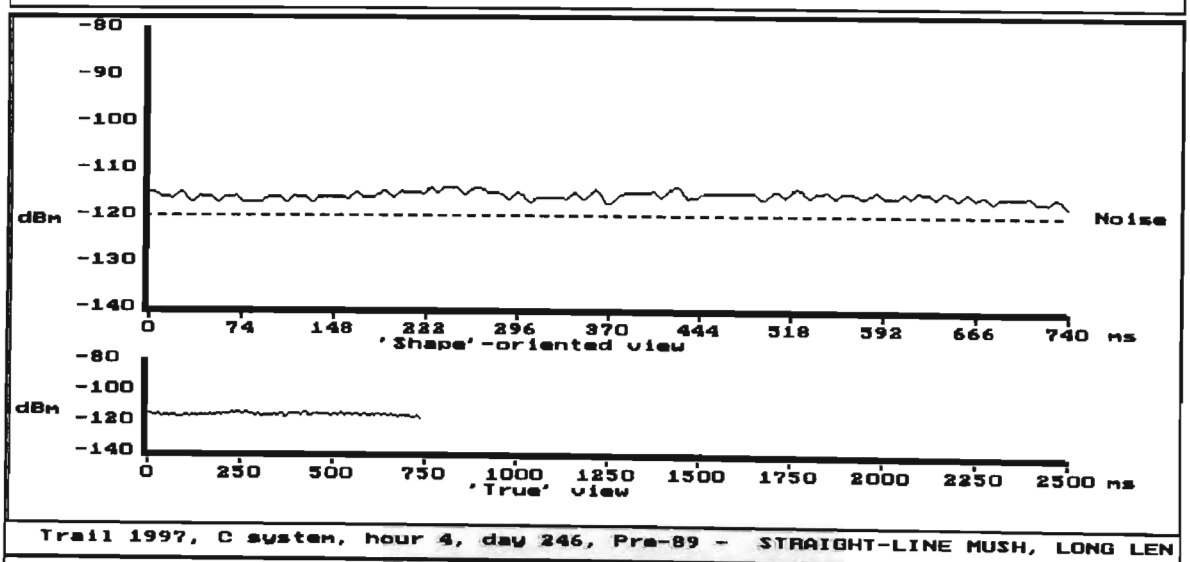


Figure 24 - Trail Type 8 : 'Straight-line Mush (Long)'

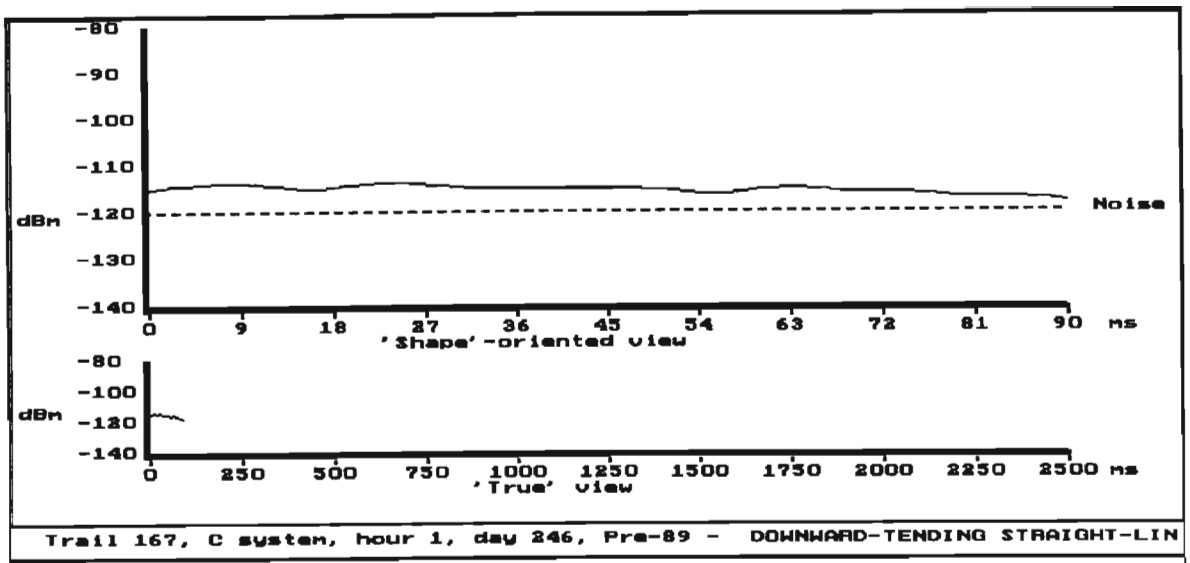


Figure 25 - Trail Type 23 : 'Downward-tending Mush'

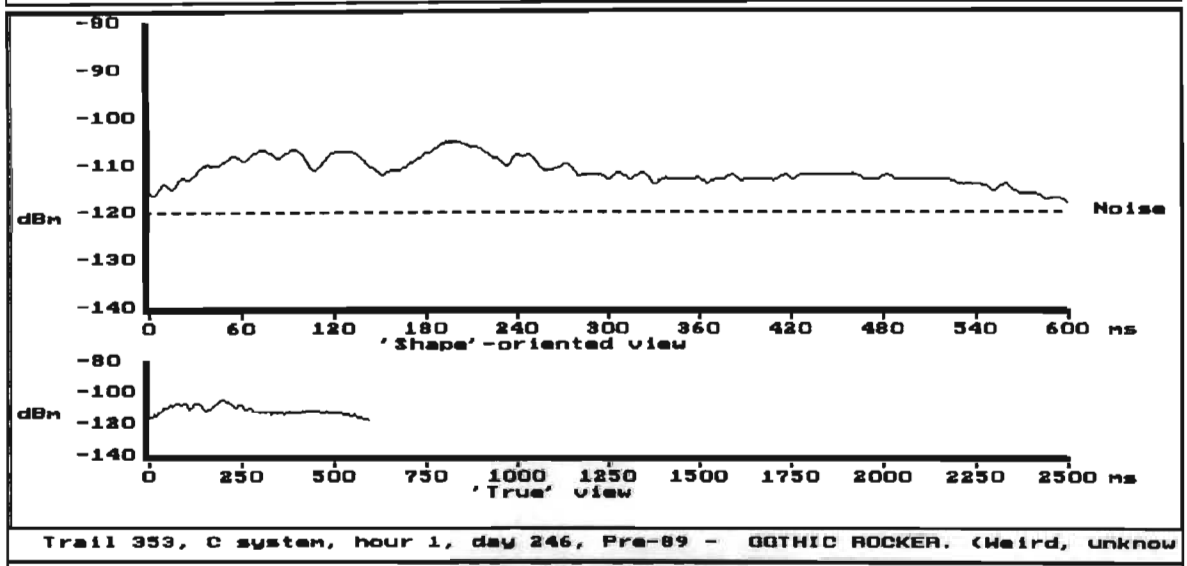


Figure 26 - Trail Type 22 : 'Gothic Rocker (Unknown type)'

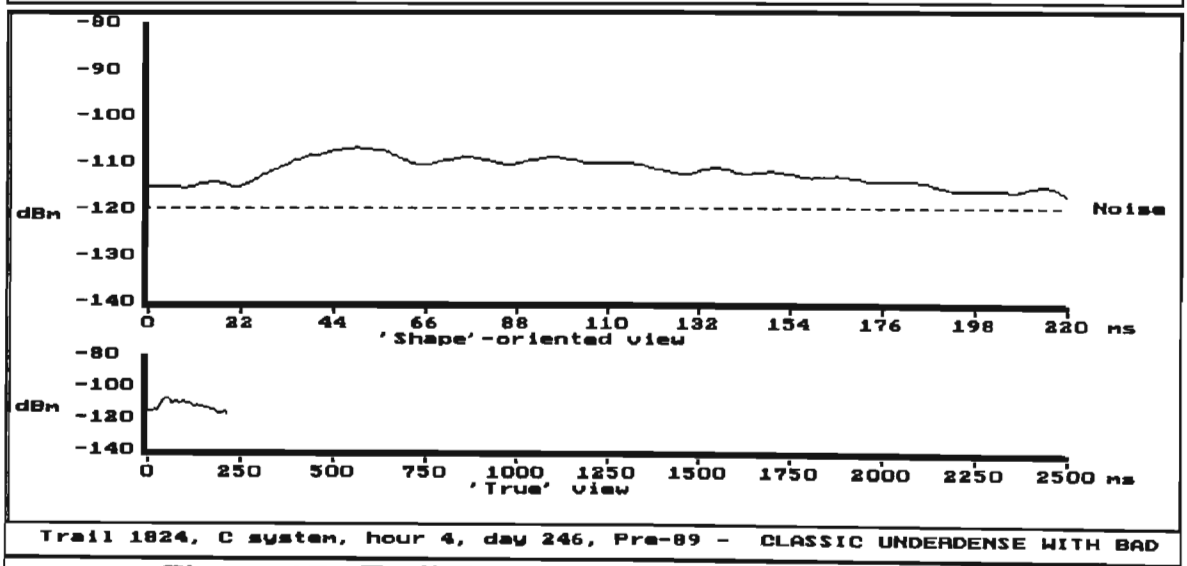
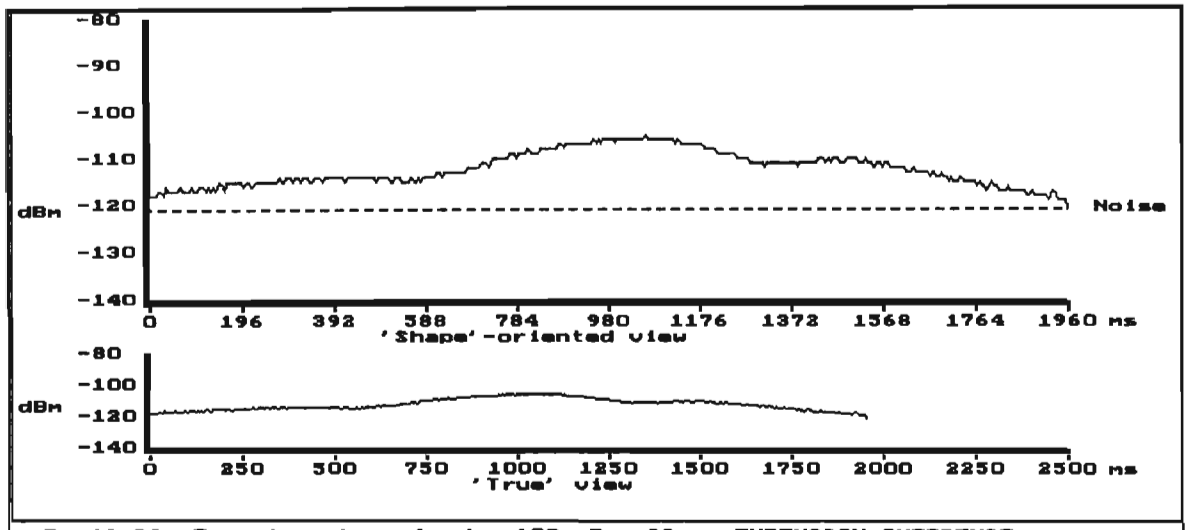
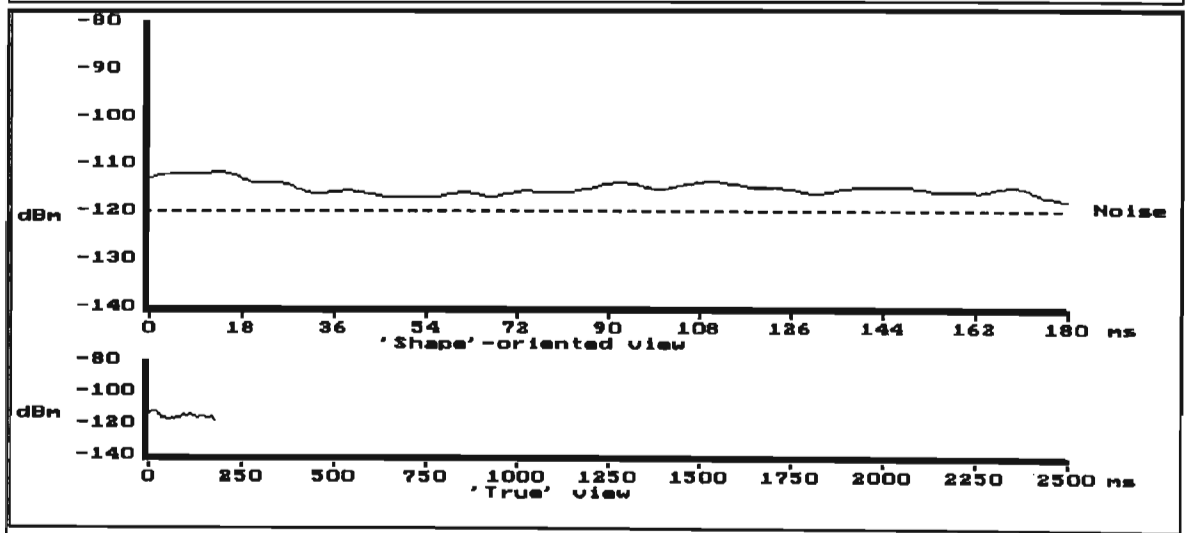


Figure 27 - Trail Type 14 : 'Classic U/D, Bad Rise'



Trail 96, E system, hour 1, day 100, Pre-89 - EXTENSION OVERDENSE.

Figure 28 - Trail Type 26 : 'Extension Overdense'



Trail 264, C system, hour 1, day 246, Pre-89 - EXTENSION MUSH.

Figure 29 - Trail Type 25 : 'Extension Mush'

The 'extension' types, which are primarily determined on the basis of their arriving within 10 ms of the previous trail, and having similar characteristics, are depicted in Figures 28 and 29. However it should be noted that their major determining feature, arrival time, does not lend itself to the duration/amplitude display. In addition, the 'Gothic Rocker' type (type 22) is shown in Figure 26. The reader should also be aware that this group contains precisely those trails which do not fit into any other classification - the 'unknowns', and that clearly there can be no such thing as a 'typical example' of a group defined as being that containing trails which fit no typical pattern.

Figures 30 to 31 below give distributions of trail type counts and durations for the 1100km midpath link, while Figure 32 shows distributions of trail type counts for the 550km link, in all cases over twenty-four hour periods (to avoid the effects of daily cyclical variations). The actual distributions given are from data captured in the period May to June, 1987. (Measurement links as described in Table 1.) The contributions of the various trail types to counts and durations in these distributions seem typical of data captured over a two-year period.

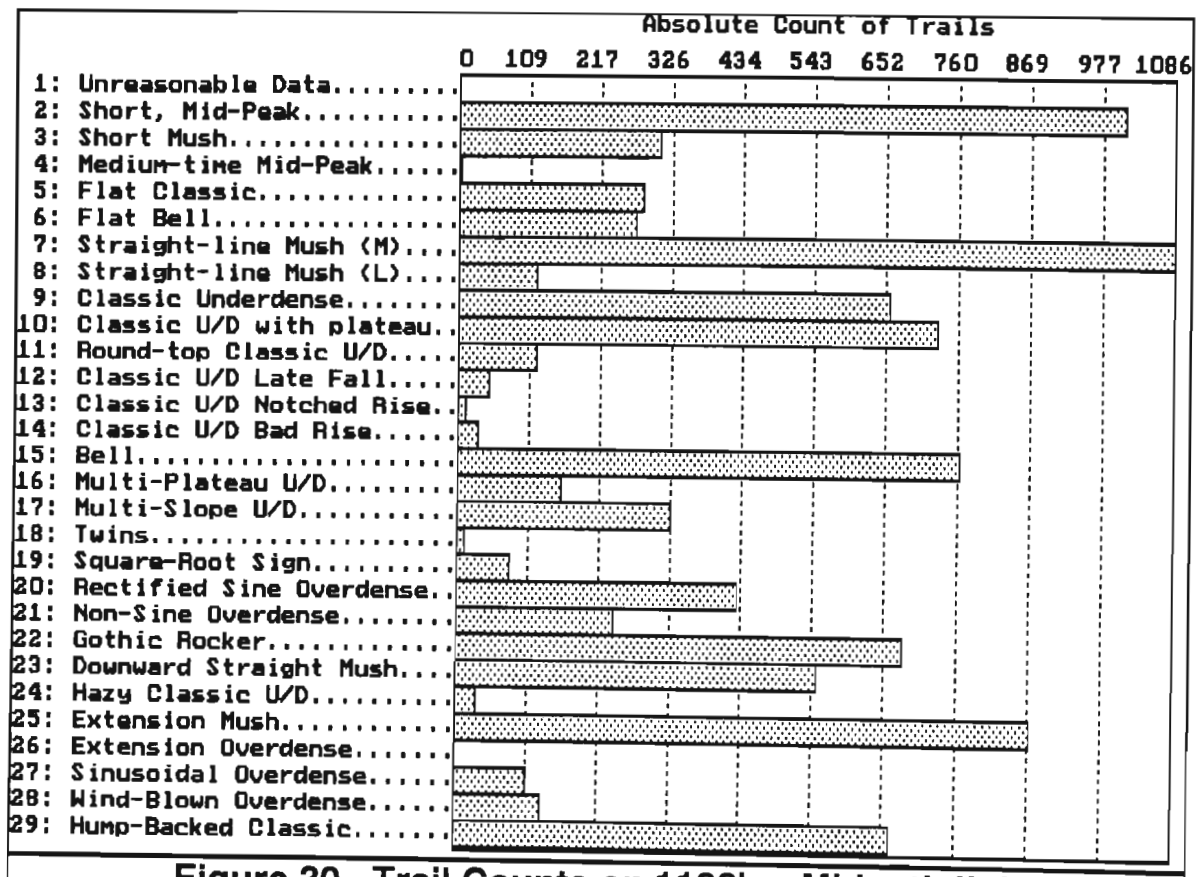


Figure 30 - Trail Counts on 1100km Midpath link

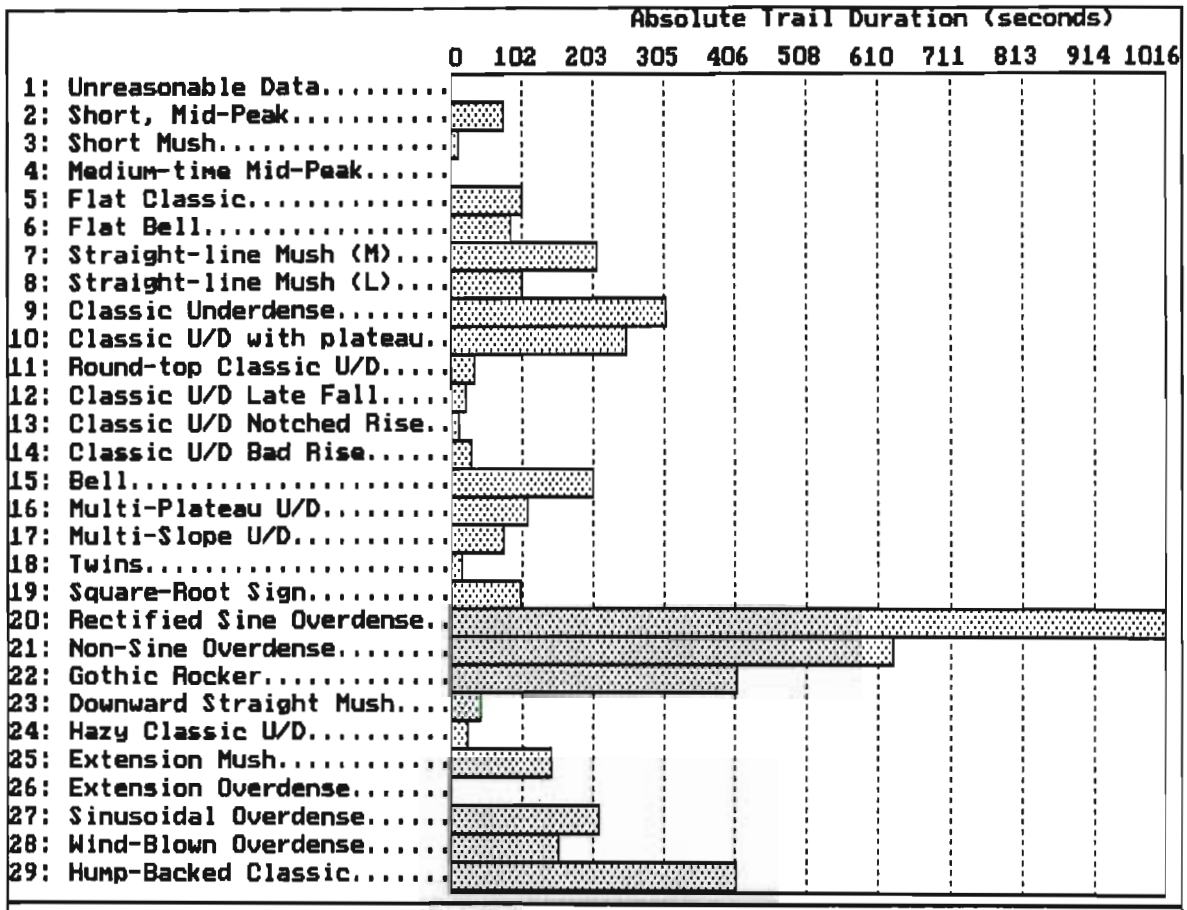


Figure 31 - Trail Durations on 1100km Link

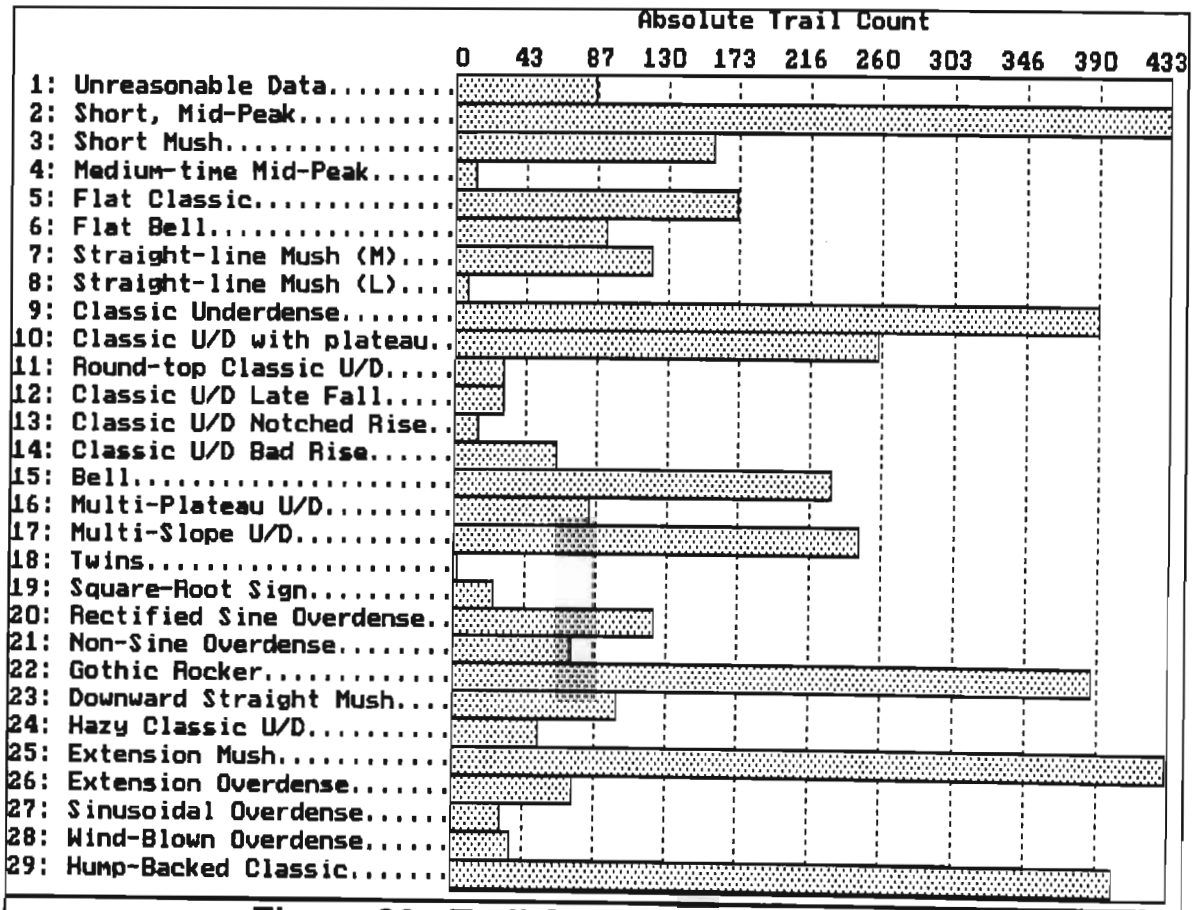


Figure 32 - Trail Counts on 550km Link

What is immediately evident in examining the count distributions is that the trail type 9, which corresponds to the classic underdense shape shown in Figure 1 is one of the more dominant of the trail types. However it is noteworthy that trail type 29, which exhibits a characteristic 'hump' in the downward slope, as shown in Figure 2, is as significant. This characteristic distortion in the classic shape could arise from a number of mechanisms, and can result in significant distortion of underdense trail statistics. Type 10, the underdense classic with plateau (see Figure 3) is another significant subset of the underdense group. This family is again distinct from the classic underdense model, and could arise from transitional underdense-overdense effects as discussed by McKinley [47] or a number of other effects.

Trail types 11 to 14 are not as significant as other variations on the classic underdense shape and so will not be discussed in detail. However, it is worth noting that the 'bad' rise types, types 13 and 14, do not have the normal smooth underdense rise (see Figure 12). Since in meteor data communication systems the time required to achieve 'lock' (modem synchronisation and data handshaking) will be affected by any distortion in the rise slope it is important to recognise these types.Other variations on the underdense shape include the 'flat classic', type 5, shown in Figure 9, whose distinguishing feature is a small amplitude range, and the two multiple time-constant types, (types 16 and 17, shown in Figures 17 and 18), which have more than one distinct slope in their fall region. The 'hazy classic', type 24, (see Figure 15), is a trail which clearly follows the underdense model but has some feature(s) which prevent its inclusion into any of the other underdense types.

The large number of distinguishable sub-types within the underdense family, together with the fact that the 'classic underdense' type 9 constitutes only a significant minority of these trails, does seem to strongly suggest that theoretical research done using only a single underdense model will be prone to a fair amount of error.

The trail types 20, 21, 26, 27 and 28 are all variations on the standard overdense trail shapes. The most significant is the 'rectified sine wave' overdense (shape shown in Figure 4) which, according to Jay Weitzen [48], probably results from multipath effects. The 'extension overdense' type results out of the fading of the overdense trail re- triggering the monitoring system and therefore being registered as an independent reflection. The 'sinusoidal overdense' (see Figure 5) does not have the distinctive sharp fades of the rectified sine type, but does have a sinusoidal waver superimposed on the normal specular overdense. The 'non-sine overdense' (see Figure 6) corresponds to overdense trails which do not exhibit any of the characteristic fading mechanisms as discussed under 'sinusoidal' and 'rectified sine' overdense trails. These would include the specular overdense trails as distinct from the non-specular as defined by Oetting [26]. The 'wind-blown overdense' type (type 28) shown in Figure 14 is typified by a basic specular overdense shape over the majority of the trail with some significant 'aberration' which is probably the result of wind distortion.

In examining the distribution in Figure 19, there are a number of significant shapes which do not resemble the classic underdense or overdense reflection shapes. For example, type 2, the 'short mid-peak' which has an almost triangular shape, as shown in Figure 16, (this seems to be Ostergaard's 'tiny' type) probably arises from reflections at heights where the diffusion time constant is small. Trail types 3,7,8,23 and 25 are various forms of 'mush' which correspond to propagation that does not resemble meteoric reflection and probably results out of non-meteoric propagation mechanisms. In particular the significant family of 'extension mush' results out of low amplitude signals breaking through the system threshold and is probably the result of ionospheric or tropospheric scatter [42]. However, they could arise from other effects, such as very low-amplitude non-specular overdense trails or sporadic-E.

Type 18, the 'twins' (see Figure 10), clearly results from two consecutive underdense trails with multi-path interference occurring during the period the trails co-exist. The

'square-root sign', type 19, is shown in Figure 13. This type has a very well-defined shape, and is a small but regular component of counts. Type 1 is used as the 'bucket' for 'unreasonable data' - reflection records having one or more values out of normal system dynamic range, which are the result of general equipment faults. 'Unknown' trail reflections (type 22) accounted for between six and ten percent of trail reflections depending on the system employed. No definable groups (shape or theory based) could be detected within this type.

The 'bell' types (types 6 and 15) resemble the parabolic shapes of sections of overdense trails and possibly result out of small parts of the overdense trails rising above the monitoring threshold. The 'flat bell' shown in Figure 8 is a low amplitude range variation on the 'bell' shown in Figure 7. These two form a significant family as can be seen in Figures 19 and 21, and if included in the generation of underdense statistics will significantly distort them.

Figure 20 shows duration contributions from the same trails whose counts are described in Figure 19. As was to be expected the various 'overdense' types, while not constituting a particularly large proportion of trail reflection counts, do account for a great deal of total reflection durations (more than half of the total). The particularly pronounced importance of type 20, the'rectified sine' type, certainly deserves further study as this has important implications for wait time and throughput data communication issues.

There seem to be five major 'super-groups' present which have sufficient duration to support meteor-burst communications (as opposed to, for example, the 'mid-peak' families). These are the underdense group, the overdense group, the 'bell' group, the 'mush' group and the unknown group.

These are supersets of individual trail types, as defined below :

- Underdense Group - types 5, 9, 10, 11, 12, 13, 14, 16, 17, 18, 24, and 29.
- Overdense Group - types 19, 20, 21, 26, 27 and 28.
- 'Bell' Group - types 6 and 15.
- 'Mush' group - types 3, 7, 8, 23 and 25.
- Unknown Group - type 22.

Figures 33 and 34 below show the percentage contributions of these groups to trail counts and duration for the 1100km link. There is an important result evident here, which has consistently been borne out both on every link used in this study and over every period of data capture on these links over a period of three years. This is that while the underdense group tends to be the biggest contributor to trail counts, it is the overdense group with its fewer, longer-duration trails which is the by far the biggest contributor to total duration. This has major implications for meteor-burst communications, as will become evident in the next chapter of this thesis.

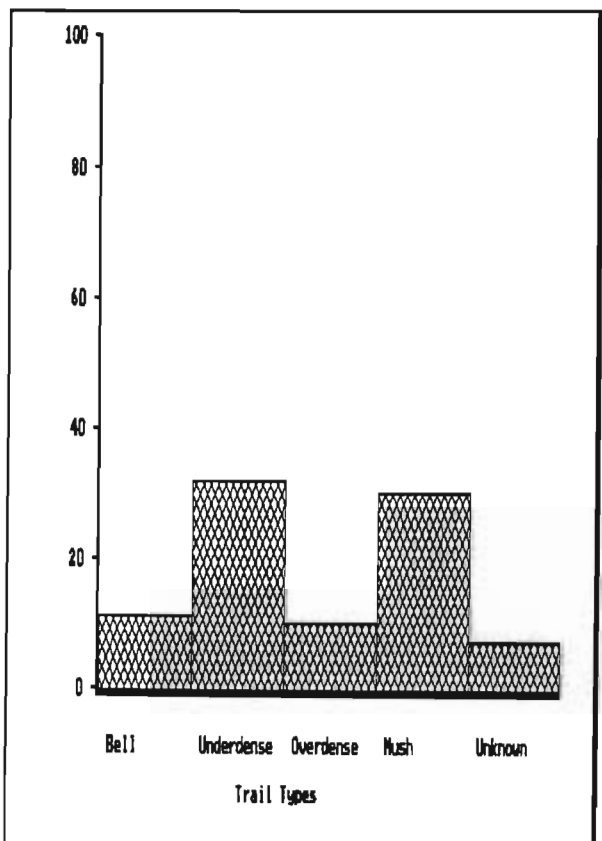


Figure 33 - Percentage Contributions to Trail Counts

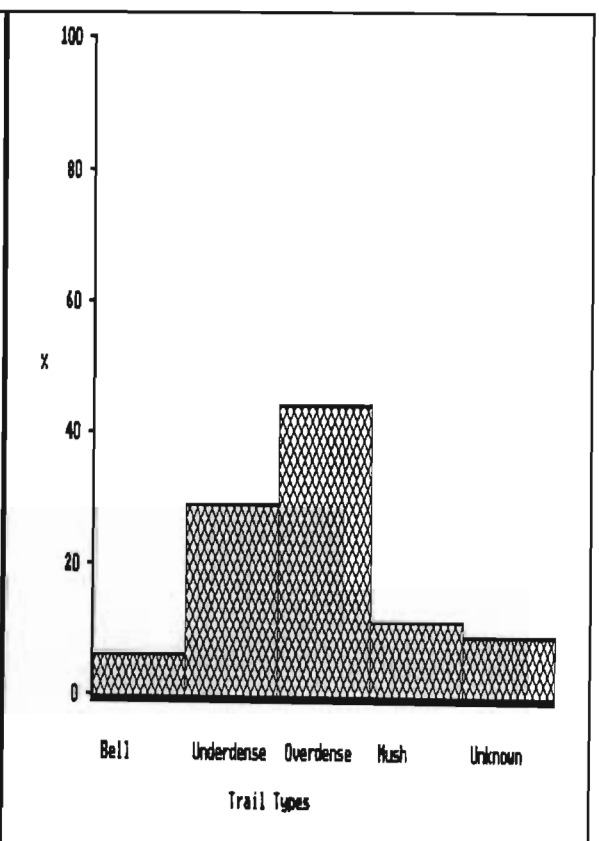


Figure 34 - Percentage Contributions to Trail Durations

2.9 Conclusion

The trail classification schema has been useful in identifying common trail shapes with distinctive characteristics which could arise from various physical mechanisms. Each trail reflection shape has been analysed, and statistics presented in order to indicate the frequency of the various types.

The analysis has indicated that further investigation into the mechanisms that produce the various characteristic shapes, from both a theoretical and a practical view, is most important since at present, apart from the classical underdense and specular ('non-sinusoidal') overdense trail reflections, very little information is available in the current literature on the various trail types.

The classification system discussed in this chapter achieved its objective in that it has identified common trail reflection shapes within the data, grouped them according to these classes, and given them names by which they may be identified. These names have only alluded to the physical mechanism in cases where such a linking has been well-established through current theory. (Such as the underdense and overdense families.) However, as the types are further refined, and the mechanisms which produce them are established, names could then be changed in order to link them in some way to the physical mechanism which produced them.

The current system recognises twenty-eight distinct types of meteor trail reflection, with only around six to ten percent of trail reflections being unclassified. The conditions, actions and rule base used appear in Appendices A, B and C respectively.

The efficiency of the system appears to be immune to changes in path length and antenna configuration, despite the fact that the overall trail reflection statistics may change quite considerably.

Classification on a PC-AT takes around two hours for ten thousand trail reflections which is significantly faster than the rate a human classifier could ever achieve.

The large number of distinct types is largely the result of the type/scan/type approach. By placing trail reflections of each type into 'buckets' which can then be scanned by a human, the system allows differences within a type to be far more easily detected than would be the case if all trail reflections of all types were considered in such a scan. Thus the system effectively aids the human classifier's understanding, so allowing the expert to teach the system what has been learnt (through the addition of new rules).

As discussed in the introduction to this chapter, the major objective was to classify those trail types which '*occur often enough to warrant seperate classification*' [23]. The eventual typing presented does, it is believed, go a long way towards achieving that objective. In addition, it has yielded some important pointers to both the physical mechanisms involved in meteor trail formation, and to the relative importance of the various trail families in terms of meteor-burst data communications. The following two chapters of this thesis will discuss further research in this latter area.

2.10 A Note on Change over Time

The TrailStar system has currently been employed on a daily basis for some three years, and has classified several million meteor trail reflections. A number of domain experts (meteor-burst researchers) have interacted with the system, and their suggestions have resulted in two major enhancements to it.

Firstly, the level of classification needed differed according to the phenomena being studied. Thus while in some cases a 'tight' exact typing of all trails was required, in others a rougher (and faster) typing was adequate. In addition the ability to be able to experiment with alternate classification schema was identified as a need. To this end the system was altered to allow for multiple available rule bases, with the user having the ability to select (or create) a base tailored to his or her needs.

Secondly, the initial system allowed for justification of decisions by the simple expedient of showing the conditions which held in the fired rule. In practical use it soon became clear that this form of 'Why?' justification was insufficient. In addition to knowing why a particular rule did fire, the users of the system also wished to know why others did not. This absence of a 'Why not?' facility was a major lack in the system - if a domain expert perceives a trail to be of one type, and the system classifies it as another, then the expert should be able to receive information as to why the trail was not classified correctly in order to rectify the problem. The system was therefore modified to include a 'Why Not?' option, which returns the text form of conditions which failed in the rules which did not trigger. Where conditions have parameters, the actual value found in the trail is given along with the failed condition. (Eg. Condition : Variance of line fit less than (0.2) :- variance found in trail is 0.28)

These two alterations constitute the only modifications to the system needed over the past three years.

3 Throughput Capacity of Meteor-Burst Communications

Abstract

Much of the recent interest focussed on data communications by meteor burst has involved the study of throughput potential of the channel, and a number of theoretical models of such communications have been derived. This chapter describes results obtained from practical monitoring systems using midpath and endpath meteor burst links. Data captured by these systems is used to simulate the throughput capacity of the links, and the results are compared with those of simulations based on the theoretical models. Results of both fixed and adaptive bit rate systems are discussed and compared. In addition, the relative contributions of the various trail families to channel capacity are discussed in some depth.

Material in this chapter has been previously published in the papers 'Observations on the Relative Importance of Different Trail Families in Meteor-Burst Communications' by SW Melville, in Proceedings of the Third South African IEEE Conference on Communications and Signal Processing, June 1990; 'Throughput Capacity of Meteor-Burst Communications' by JD Larsen, SW Melville, RS Mawrey, RY Letschert and WD Goddard, in Transactions of the SAIEE, Vol 81, No 3, September 1990; and 'Adaptive Data Rate Capacity of Meteor-Burst Communications by JD Larsen, SW Melville, and RS Mawrey, in Conference Record of the 1990 IEEE Military Communications Conference, October 1990.

3.1 Introduction

The data throughput capacity of the meteor burst channel has recently received attention in the literature. (See [19, 20, 24, 49, 50, 51].) Milstein [24] demonstrated the feasibility of meteor burst communication using a simple model of the meteor channel. Weitzen [20, 49] developed a theory to estimate the data throughput of a

meteor channel using either fixed or adaptive data rates and showed that significant improvements in data throughput could be obtained by using adaptive data rates. Abel [19] developed expressions to calculate the theoretical data capacity of individual underdense and overdense trails.

In order to test the validity of the theoretical predictions, the data communication capacity has been calculated over various meteor burst links. These calculations were based on actual recorded meteor trails as opposed to theoretical models. The method of data capture and initial processing of these reflection recordings has been described by Melville [52] and Mawrey [42]. The ideal data throughput performance of the meteor burst channels has been investigated for both fixed and continually varying (adaptive) data rates. To test the validity of the ideal results in practice, half-duplex fixed data rate communication incorporating practical overheads such as propagation delay, handshaking and error detection codes was also simulated and compared to the ideal fixed rate case.

Due to the cyclical daily fluctuations in the arrival rate of meteors all simulations were performed over 24-hour periods. Importantly, results are based on throughput optimisation as opposed to wait time optimisation. Assuming that the message size is relatively small in relation to the total feasible throughput over the 24-hour period, throughput optimisation and minimising message delay are typically conflicting objectives. In the former case a high data rate which ensured greater utilisation of the larger trails would be preferred, whereas the latter would normally require a lower data rate to enable a greater number of trails to be used.

The relative communications importance of overdense trails as compared to underdense trails is of considerable interest for any successful modelling of the meteor burst environment. In this chapter the overdense group is defined by those trails classified as 'rectified sine overdense', 'sinusoidal overdense', 'square root sign',

'non-sine overdense', 'extension overdense' and 'wind-blown overdense'; and the underdense group by the families 'classic underdense', 'flat classic', 'round-top underdense', 'classic underdense with plateau', 'classic underdense with late fall', 'classic underdense with notched rise', 'classic underdense with bad rise', 'multi-plateau underdense', 'multi-slope underdense', 'twins', 'hump-backed classic' and 'hazy classic'; as defined by the classification schema described in Melville [52] and in Chapter 2 of this thesis.

A final emphasis in this chapter will be the investigation of the potential of systems which are bit-rate adaptive. Adaptive systems are the subject of a great deal of current research (see [19, 20, 24, 50, 51]) and the ability to compare performance of fixed and adaptive rates is highly relevant to the design of adaptive systems.

3.2 Measurement Links

In addition to the two midpath measurement links described in Table I, this study also considered the throughput capacity of an endpath link, as described in Table 2 below. Again, the antennae specified are horizontally polarised unless otherwise mentioned.

Link	550km Endpath
Tx Site	Pretoria (26°S, 28°E)
Rx Site	Durban (30°S, 31°E)
Tx Antenna	11-element Yagi (12 dBi)
Tx Antenna Height	9 m
Rx Antenna	3-element Yagi (8 dBi)
Rx Antenna Height	0.5 m
Rx Elevation	90°

TABLE 2 - 550km Endpath Measurement Link

3.3 Meteor-Burst Channel Capacity

The capacity of the meteor burst channel has been estimated by Weitzen [20, 49]. The model used by Weitzen considered three theoretical trail families, namely underdense, overdense and modified overdense trails. Using measured data and dividing trails into more specific trail families, using the work of Melville [42], a greater insight into the performance of real meteor burst channels has been obtained.

The ideal data throughput performance of the meteor burst channels has been investigated for both fixed and continually varying or adaptive data rates based on received power measurements taken over real meteor burst links. It is shown in [19] that the time varying power received during a particular meteor reflection, $P_r(t)$, may be related to the maximum possible time varying data rate, $r(t)$, over the channel by the following expression:

$$r(t) = \frac{P_r(t)}{N_o\left[\frac{E_b}{N_o}\right]_{req}} \qquad \text{(Equation 1)}$$

where :

$P_r(t)$ is the received power,

N_o is the received noise power spectral density, and

$\left[\frac{E_b}{N_o}\right]_{req}$ is the ratio of received energy per bit to noise power spectral density required by a modem for a specified bit error rate and type of modulation.

For a particular fixed data rate, r_{op}, data throughput occurs when $r(t) \geq r_{op}$ and in the case of adaptive data rate communication, data throughput occurs at a time varying rate $r(t)$.

Software was written to calculate the throughput characteristics of the measured data using both fixed and adaptive data rates. Due to the discrete nature of the measured data only certain data rates were considered for a particular signal-to-noise requirement, $(E_b/N_o)_{req}$. A signal-to-noise ratio requirement of 9 dB was chosen as a typical value based on error rate performance for various types of modulation (see [11]). Note that, as in all the simulations described in Chapters 3 and 4, bandwidth is normalised. Thus while the work **projects** what would happen at a variety of data rates based on the signal, noise and duration of trails captured on a 2 kHz monitoring system, data rates are not limited by the **monitoring system's** parameters.

3.4 Fixed Data Rate

Using theoretical techniques, Weitzen [49], presented plots of throughput for fixed operating rates assuming that all trails were underdense, overdense or modified overdense, the latter family being used to approximate the effects of high-altitude winds on trail decay.

Figures 35 through 37 are plots of Weitzen's results for the theoretical meteor link described in [49].

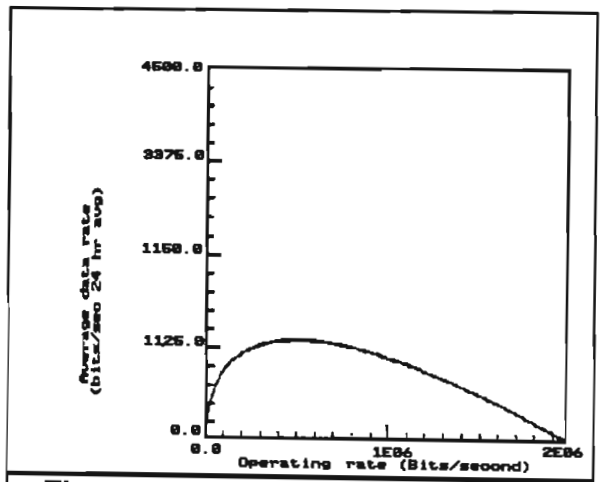


Figure 35 - Weitzen's Underdense Model

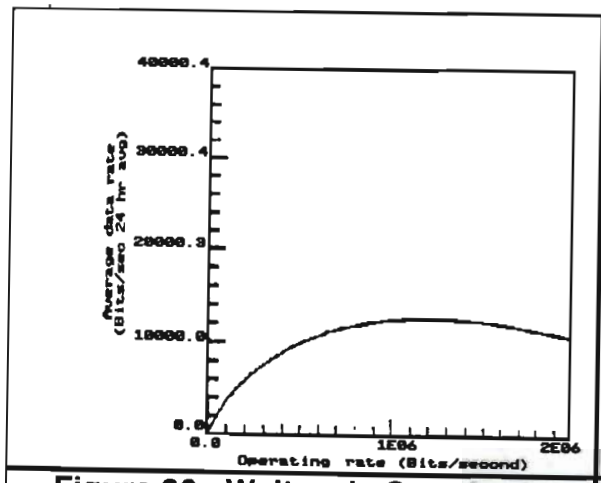


Figure 36 - Weitzen's Overdense Model

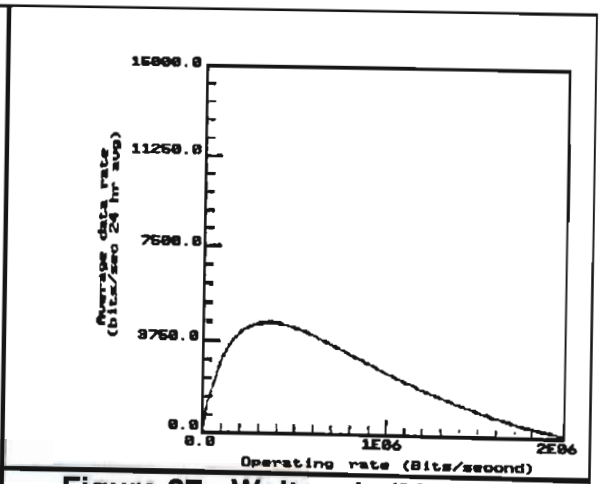


Figure 37 - Weitzen's 'Modified Overdense' Model

The fixed data rate software calculates the fraction of time, t_{op}, that communication is possible using a fixed operating rate, r_{op}, for a user-defined signal-to-noise ratio, $(E_b/N_o)_{req}$, and for user-defined trail families. The data throughput at rate, r_{op}, is the product of r_{op} and t_{op}. The algorithm is performed by reading the digitised samples of $P_r(t)$ and N_o, calculating $r(t)$, and adding the time interval at t_{op} to all values of r_{op} for which $r_{op} \geq r(t)$. The program output consists of plots of r_{op} versus t_{op} or r_{op} versus data throughput at r_{op}.

As a comparison to Weitzen's predictions the fixed data rate software was used to calculate the throughput using fixed operating rates on the 1100km midpath link. Approximately ten thousand trails were considered. Plots of simulations performed over a day using underdense, overdense and all trail families is given in Figure 38. The absolute magnitudes of results may not be compared due to the differences between Weitzen's theoretical link and the 1100km midpath link, however, overall trends may be compared.

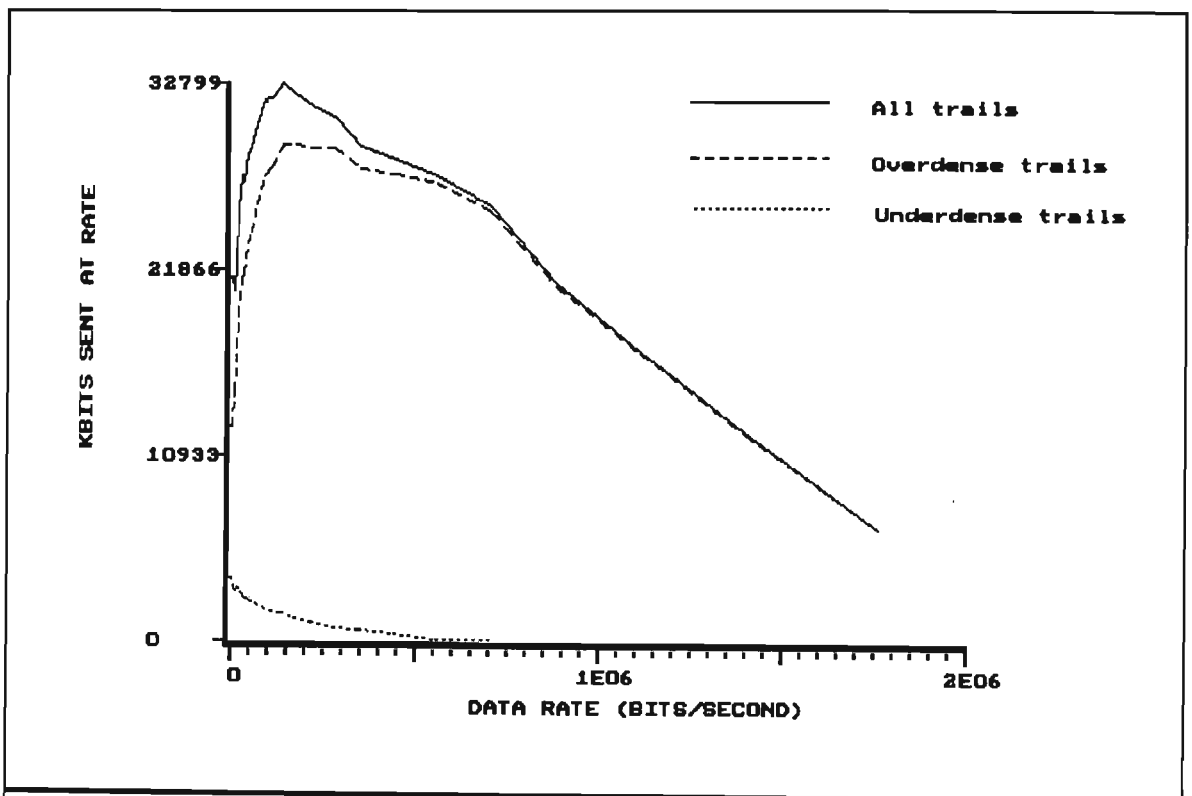


Figure 38 - Fixed Rate Throughput over 1100km Midpath Link

As mentioned by Weitzen [3], his underdense modelling is inaccurate because it was assumed that all trails followed underdense characteristics, when in practice trails with electron line densities greater than approximately 10^{14} electrons per metre exhibit overdense characteristics. This invalid assumption would explain the discrepancies between the underdense model shown in Figure 35 and the observed results shown in Figure 38, with the observed results indicating a far sharper drop in capacity as data rate increases than predicted by the model.

Weitzen's overdense model, shown in Figure 36 does not have the pronounced drop in capacity after peak found in Figure 38. This is probably the result of the model not catering for the pronounced wind effects on overdense trails (the true 'specular overdense' trail without any wind distortions was shown in [52] to occur so rarely as to be insignificant).

The 'modified overdense' model shown in Figure 37 does consider wind distortion, and it is clear that this model is far more compatible with the observed results. In both the model and the observed results there is an early rise to peak capacity at a rate between 100 and 400 kilobits. The model has a less pronounced drop after this peak capacity is reached, but this discrepancy could as easily be explained by the limited periods used for the observed results as by the presence of any inaccuracy in the model.

Figures 39 and 40 show the results of throughput simulations of the 550km midpath and endpath link. The results of the simulations reveal that the data throughput is dominated by the overdense family of trails, as was the case with the 1100km link.

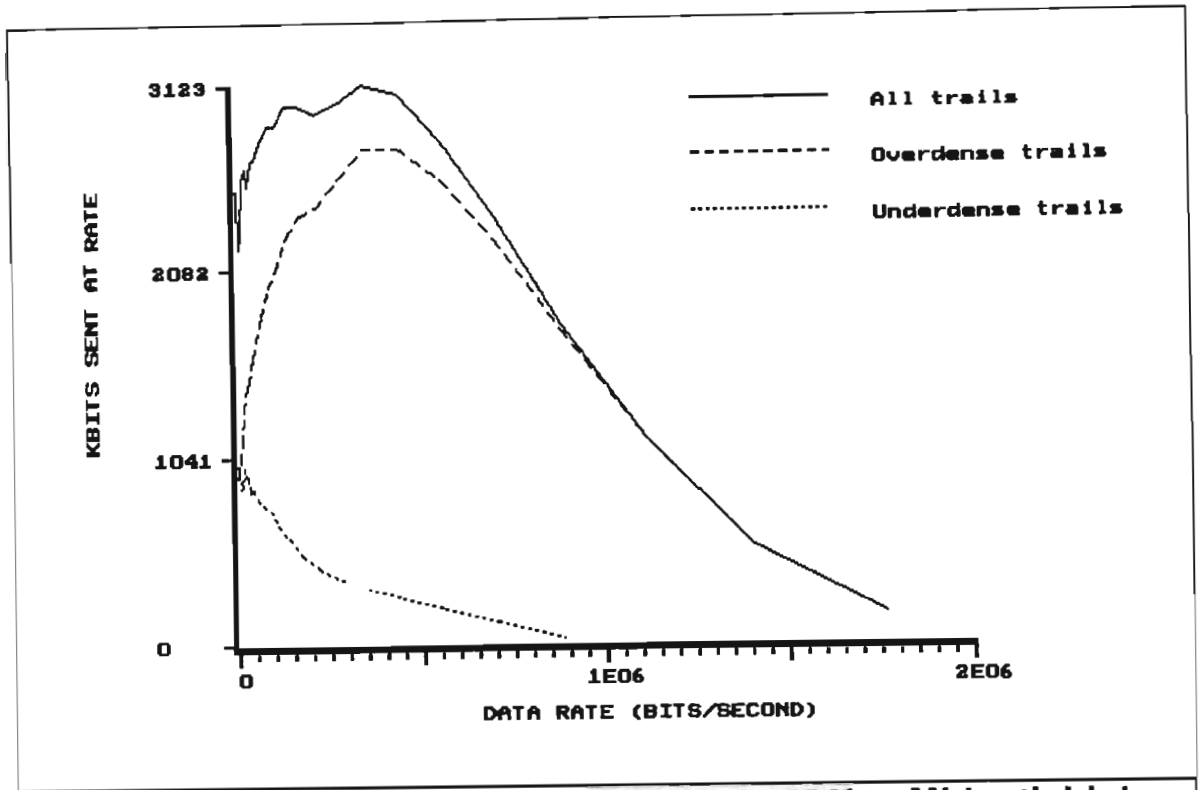


Figure 39 - Fixed Rate Throughput over 550km Midpath Link

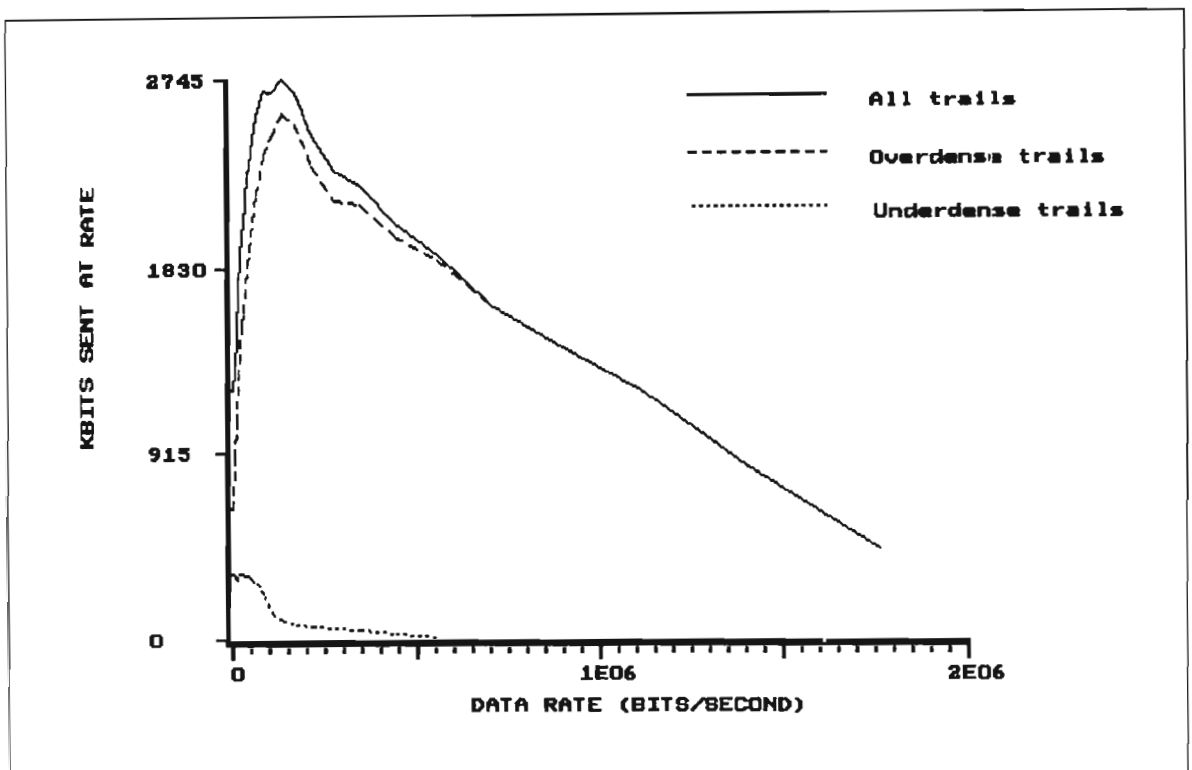


Figure 40 - Fixed Rate Throughput over 550km Endpath Link

Simulations of the 550km midpath link for three consecutive days, given in Figure 41, show that there are significant daily variations in fixed throughput characteristics which emphasizes the need to make use of adaptive data rates.

Figure 41 -Consecutive Days Throughput, 550km Midpath Link

3.5 Adaptive Data Rate

As shown by Formula 1, making optimal use of a meteor trail requires the data rate to adapt to the received signal-to-noise ratio. Using adaptive data rates, Weitzen [20, 49], estimated the number of seconds per day, $\Gamma(r)$, for which a meteor channel would support a data rate between r and $r+\Delta r$. Figures 42 to 44 are weighted rate density functions of $r.\Gamma(r)$ versus r for theoretical underdense, overdense and modified overdense models. A value of Δr equal to 5000 bits per second was used by Weitzen [49].

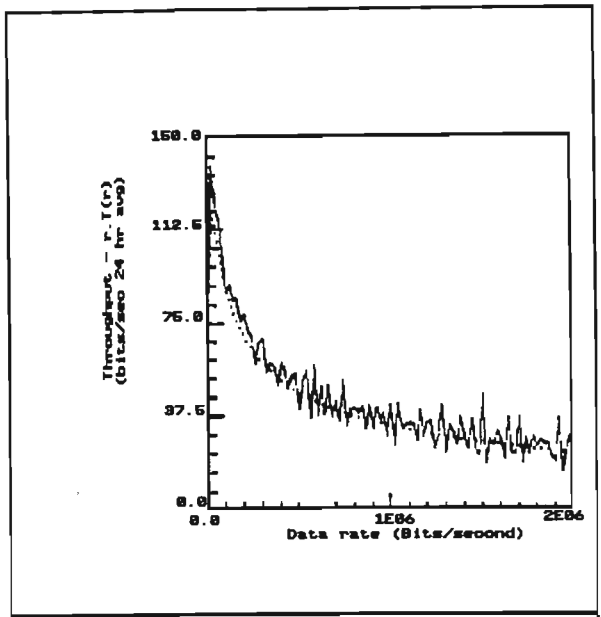


Figure 42 - Theoretical Underdense Model

Weitzen's weighted rate density function for the theoretical underdense model [49]. Solid line = results of simulations, dotted line = closed form approximation.

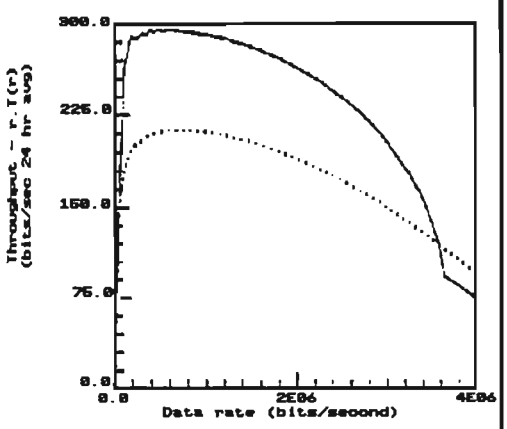


Figure 43 - Theoretical Overdense Model

Weitzen's weighted rate density function for the theoretical overdense model [49]. Solid line = results of simulations, dotted line = closed form approximation.

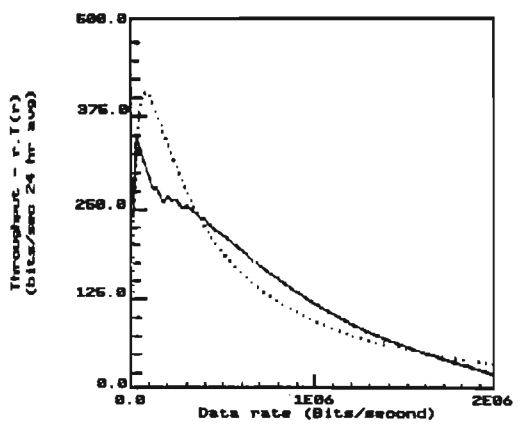


Figure 44 - Theoretical 'Modified' Model

Weitzen's weighted rate density function for the theoretical 'modified' model [49]. Solid line = results of simulations, dotted line = closed form approximation.

Software was written to calculate weighted rate density functions for the measured data. Each measured sample is examined by the software to calculate the data rate it would support. Then $\Gamma(r)$ is incremented by the sampling period for a data rate between r and $r+\Delta r$. Due to digitisation of the measured data into decibel steps the increment in the calculated data rate, dr, was not constant and increased exponentially with data rate, r. To overcome the variation in the increment in data rate, and to ensure compatibility with Weitzen's results, results were normalised to steps of 5000 bits per second. The output of the program consists of weighted rate density functions in histogram form, where the width and height of each bar is determined by Δr and $r.\Gamma(r)$ respectively.

Figures 45 to 47 are plots of the weighted rate density function for the 1100km midpath link using underdense, overdense and all trail families respectively. As in the fixed data rate case it is clear that the throughput is dominated by the overdense trail family. A comparison between the theoretical and measured weighted rate density functions reveals significant differences. As in the case of fixed data rates it is the theoretical modified overdense model that is the closest approximation to the measured data. The correlation between the measured and predicted results is, however, poor, with the weighted rate density function having a different form and a greater concentration of throughput at the lower data rates. It should be noted that the measured throughput does not appear to approach zero as the data rate approaches zero due to the digitisation of the data rate into 5 kbps steps.

Graphs of weighted rate density functions for the 550km midpath and endpath links are given in Figures 48 to 53. A comparison between the underdense and overdense contributions reveals that the overdense trails dominate the data throughput. The similarity in the shape of the weighted rate density functions of the three links shows that the form of the weighted rate density functions is not highly dependent on differences amongst link configurations.

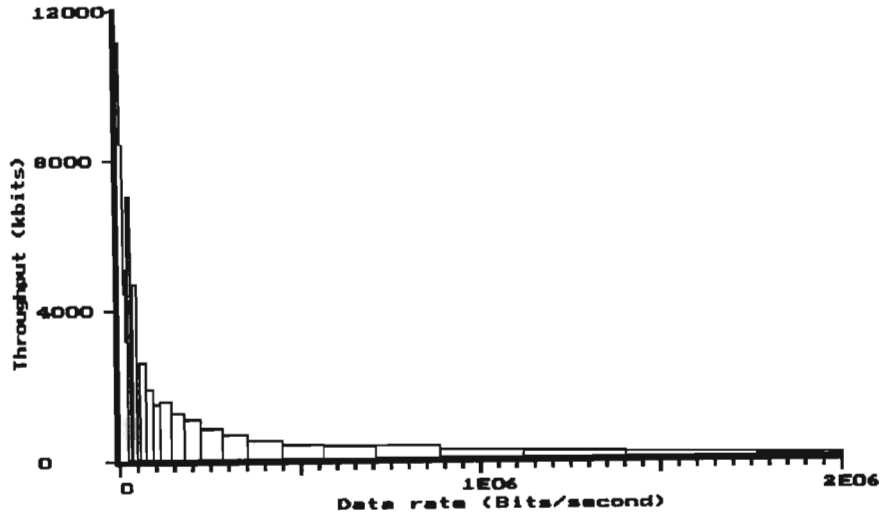


Figure 45 - All Trails, 1100km Midpath Link

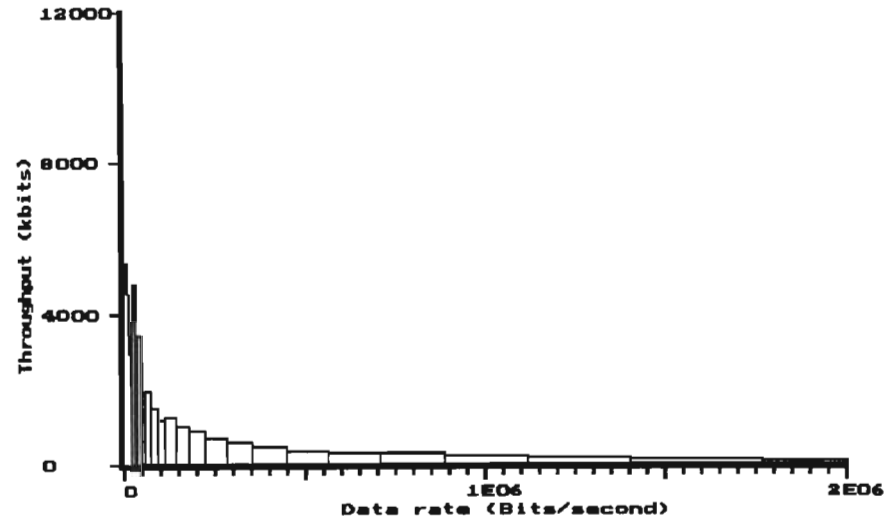


Figure 46 - Overdense Trails, 1100km Midpath Link

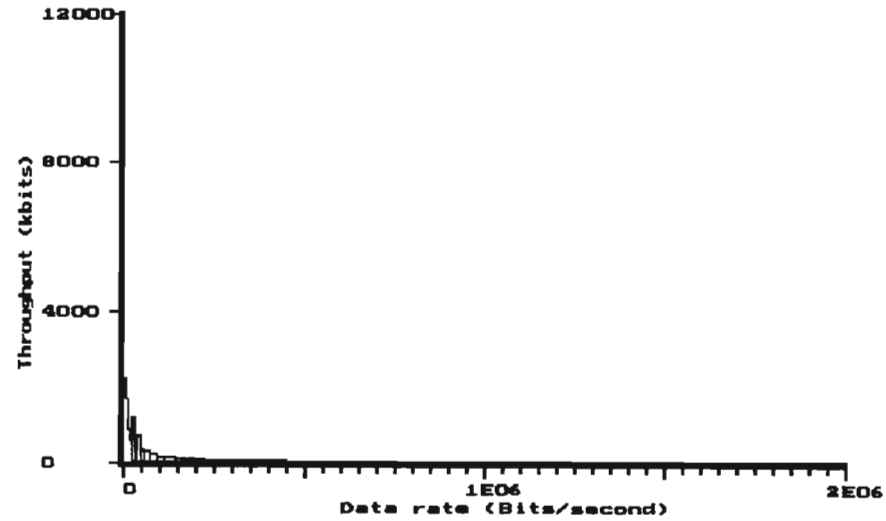


Figure 47 - Underdense Trail, 1100km Midpath Link

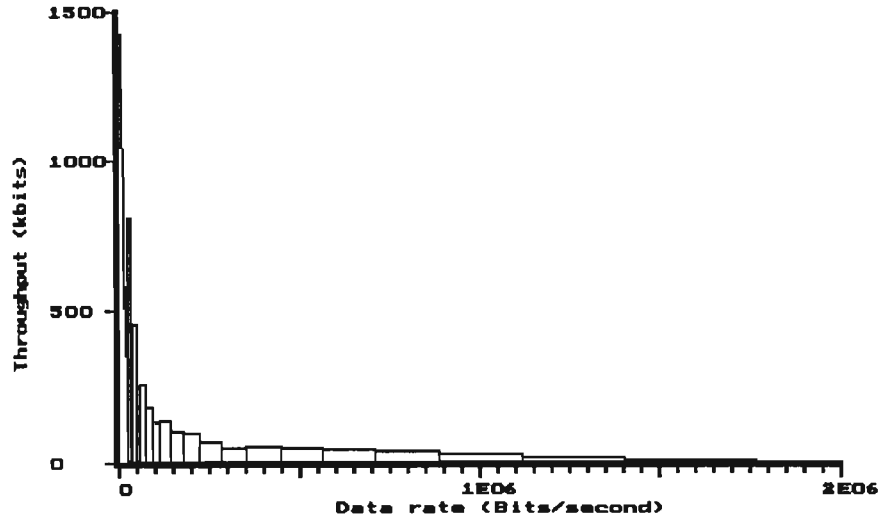


Figure 48 - All Trails, 550km Midpath Link

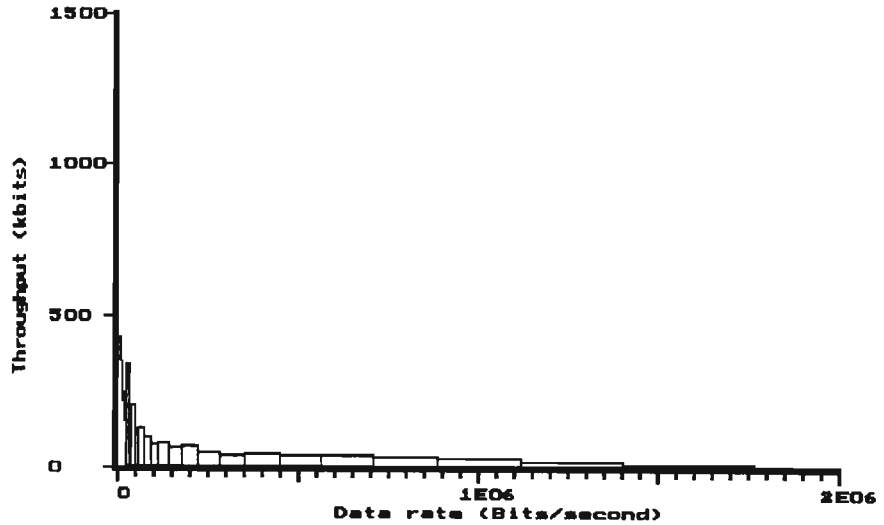


Figure 49 - Overdense Trails, 550km Midpath Link

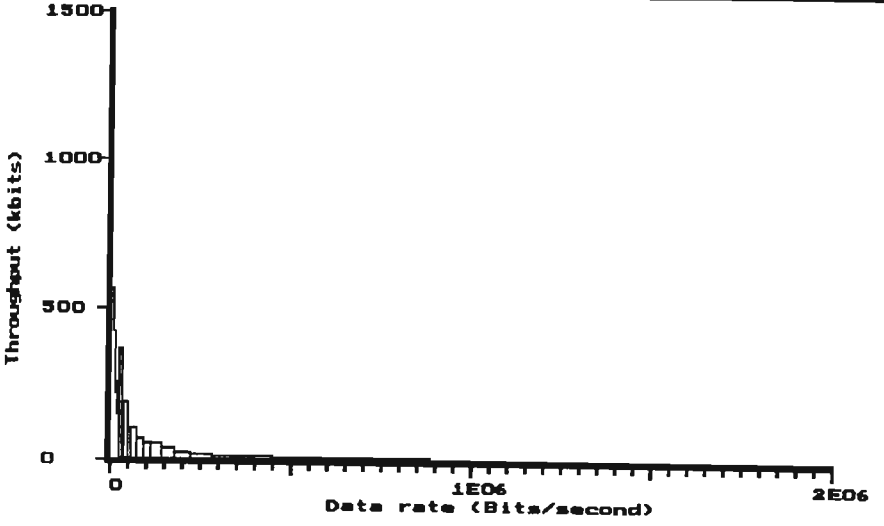


Figure 50 - Underdense Trails, 550km Midpath Link

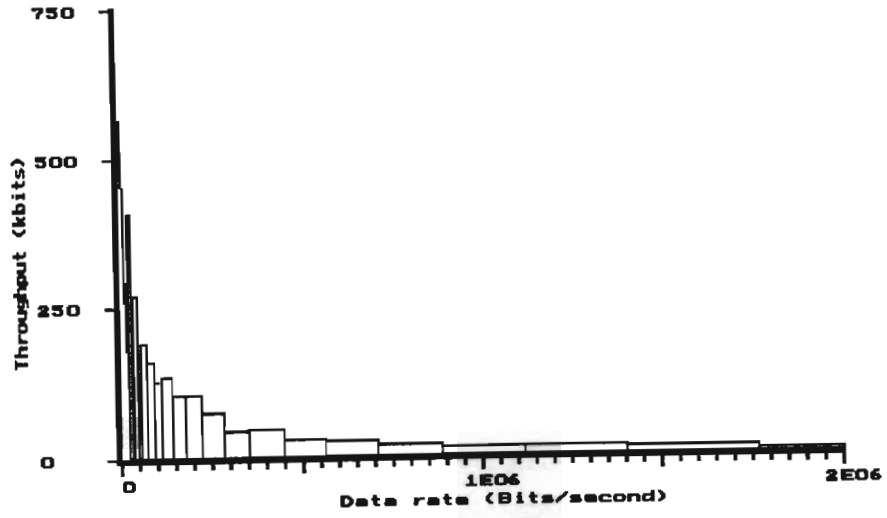


Figure 51 - All Trails, 550km Endpath Link

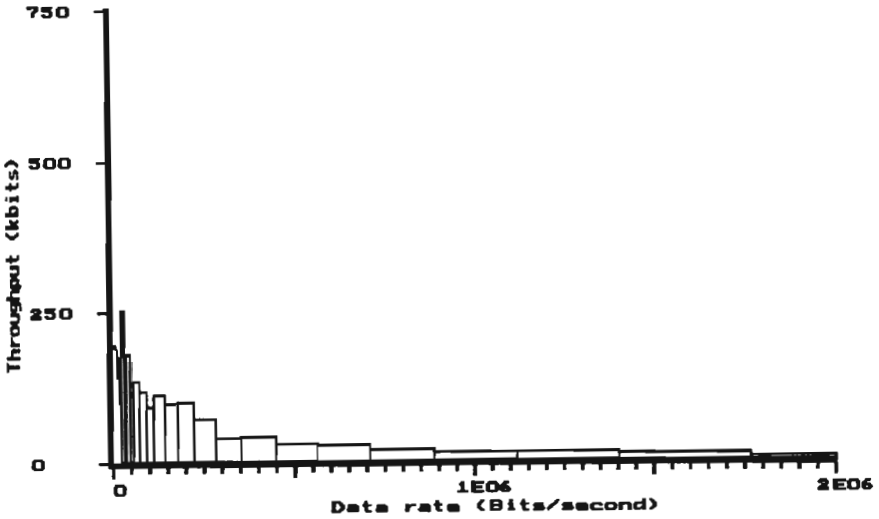


Figure 52 - Overdense Trails, 550km Endpath Link

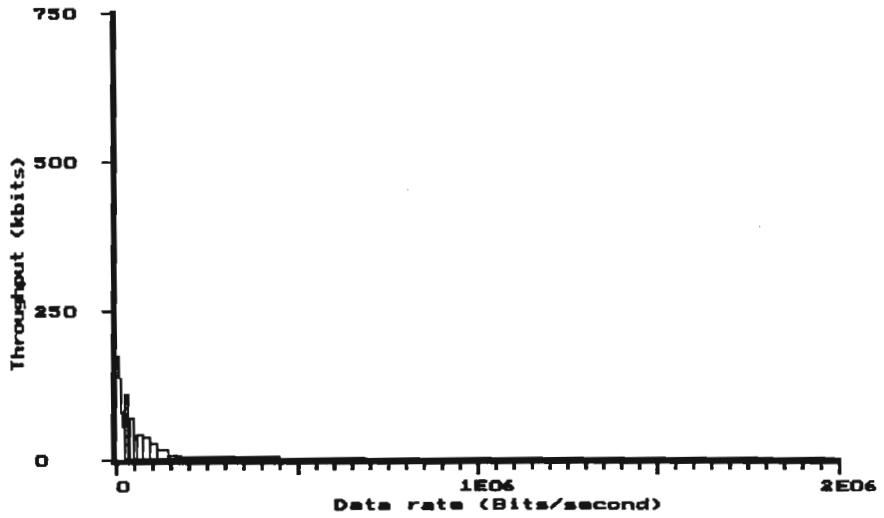


Figure 53 - Underdense Trails, 550km Endpath Link

3.6 Fixed versus Adaptive Throughput

The relative gains in using adaptive rates as opposed to fixed rates were in the order of three hundred percent. The exact figures were: 1100km midpath system 358%, 550km midpath system 371% and 550km endpath system 298%.

Figure 54 is a predicted comparison between fixed data rate throughput and limited adaptive data rate throughput across data rates varying up to two megabits per second, using the 'modified overdense' model, as derived by Weitzen [49]. Limited adaptive data rate is defined as making use of data rate adaptivity for rates r and operating at rate R_f for $r \geq R_f$.

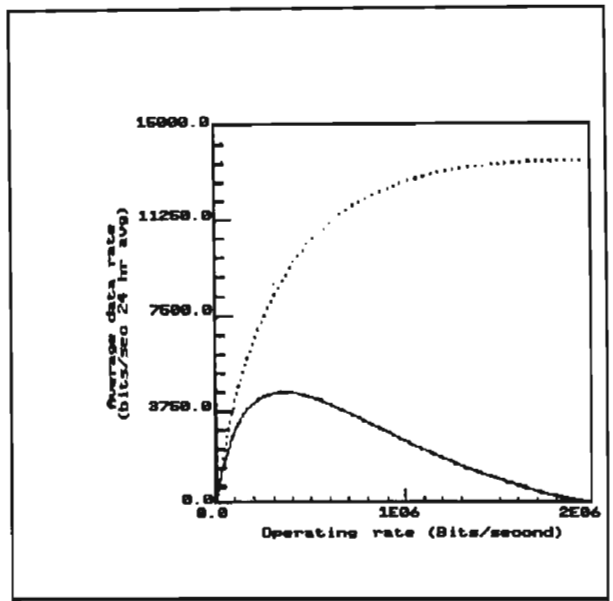


Figure 54 - Weitzen's Predicted Fixed vs Adaptive Results

Figures 55 to 57 below show results obtained for the three links used in this study.

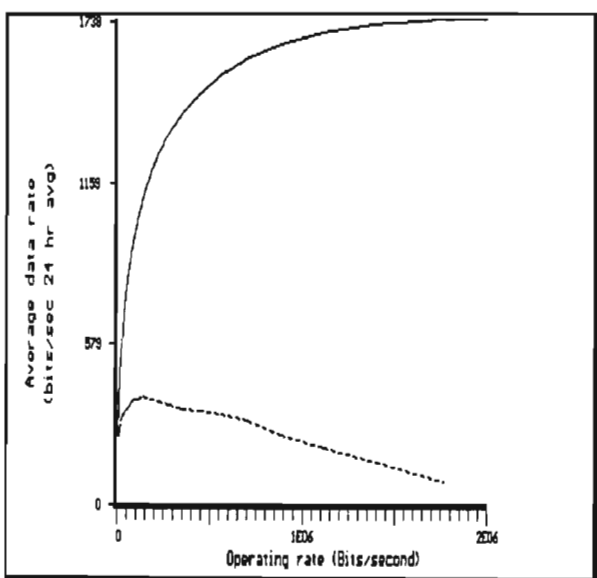


Figure 55 - Fixed vs Adaptive, 1100km midpath link

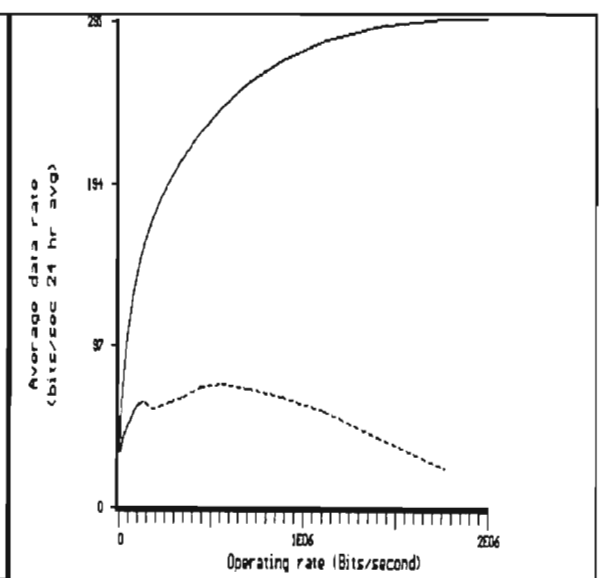


Figure 56 - Fixed vs Adaptive, 550km midpath link

The comparison between fixed and limited adaptive throughput shows that there are significant gains to be had in using the latter technique. These gains are most pronounced at the lower operating rates, indicating that at these rates small increases in adaptive rate capacity of modems can yield large gains in throughput. In general the measured results conform well with those predicted by Weitzen (Figure 54), although they do indicate a greater increase in adaptive gain at the lower rates.

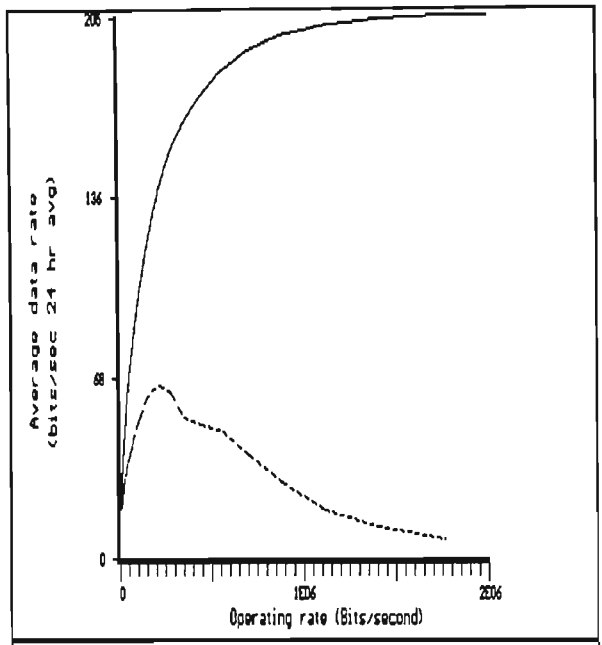


Figure 57 - Fixed vs Adaptive, 550km endpath link

In order to compare limited adaptive gain as operating rate increases across all three systems the results were normalised, and are presented in Figure 58 below. The fact that similar results were obtained for all three links indicates that adaptive gains are not highly sensitive to variations in link parameters.

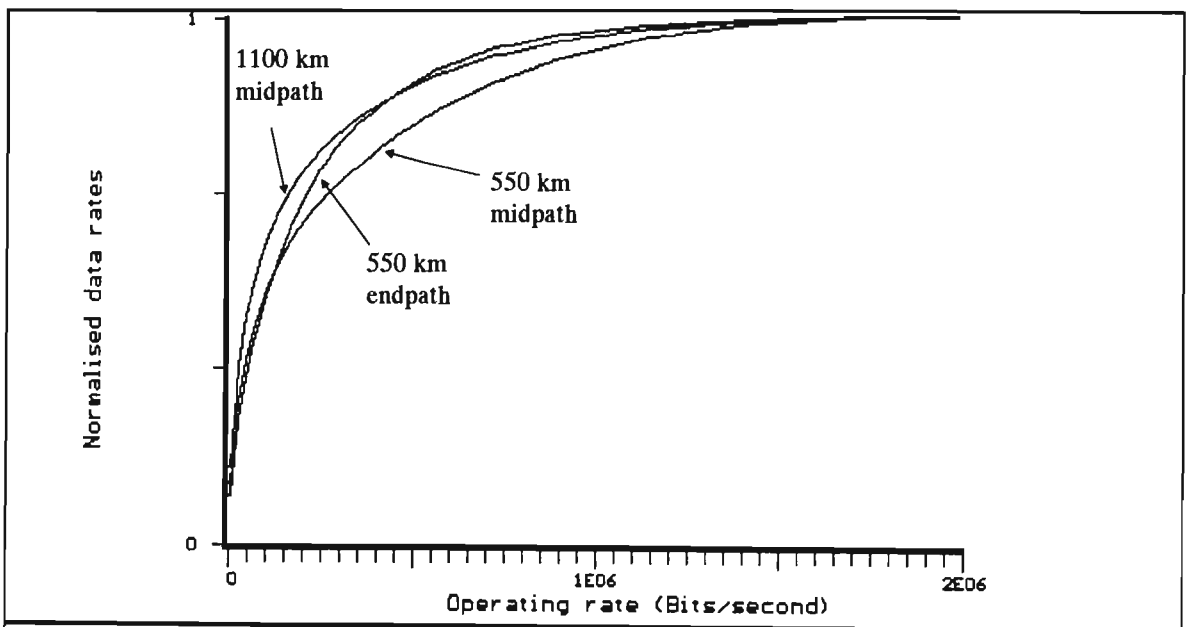


Figure 58 - Normalised Adaptive Gains Across Links

3.7 Data Communication Simulation

The preceding results have been based on the assumption of data communication with completely variable block sizes (from a single bit to an infinite number of bits per block), no control or validation overhead, no turnaround delay, and no propagation delay. As this obviously does not hold in a real-world situation, there was a possibility of some of the results presented being misleading. (One expects a high proportion of low-density trails, for example, due to the inverse relationship between trail density and number of trails having that density, first mentioned by Lovell in 1947 [5]. Many of these trails would have insufficient duration to support any communication whatsoever in a real-world environment, but their sheer numbers could seriously skew the 'no-delay' results, particularly insofar as the lower data rates are concerned.)

A software simulation of a real environment case was necessary to give some indication of the degree of error for practical communications based on the 'no-delay' results earlier presented. A simple probe-send-acknowledge system with block CRC checks was simulated, as illustrated in Figure 59.

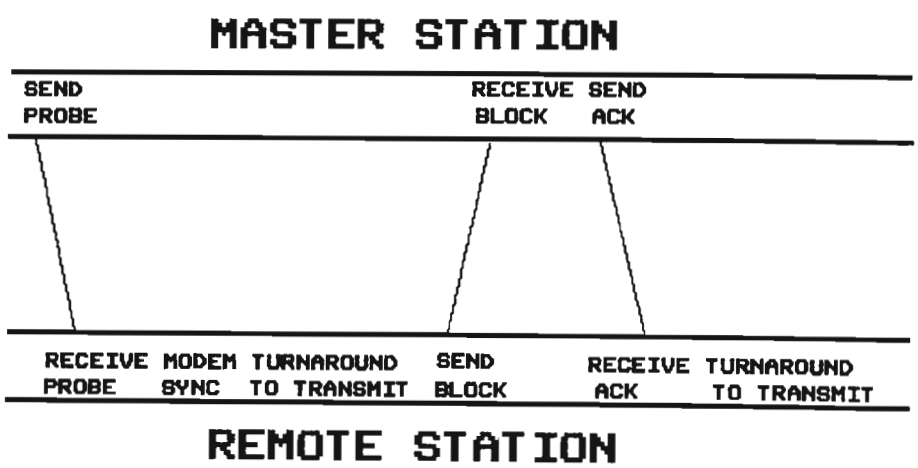


Figure 59 - Simulation Diagram

The simulation envisages half-duplex communication between a central station and a remote station, as would be the case in most of the current meteor burst installations. (The Alaska weather network [25] or the SNOTEL system [35] as examples.) It also assumes that communication will be at some fixed rate, rather than an adaptive rate. While there is a great deal of work currently being done in the investigation of adaptive-rate communications in meteor burst, most current real-world systems are fixed-rate systems.

In the simulation, the user has control over a number of parameters. In the results presented here, parameters were set as follows :

Minimum bit rate	0	kbps
Maximum bit rate	500	kbps
Minimum block size	80	bits
Maximum block size	64000	bits
Path length	550/ 1100	km
Bandwidth	2000	Hz
Probe length	10	bits
CRC length	8	bits
ACK length	8	bits
Modem synchronisation time	2	ms
Turn on error rate	0.001	
Turn off error rate	0.01	
Master power	400	Watts
Remote power	400	Watts
Switching time, transmit-receive	1	ms
Switching time, receive-transmit	2	ms

Table 3 - System and Link Parameters

Note that power and bandwidth define only the monitoring system on which the trails to be processed were captured, and in no ways are used as limiting factors on possible bit rates. Differential PSK modulation is assumed for the simulation, and thus the formula $Error\ Probability = \frac{1}{2} e^{-\frac{E_s}{N}}$ applies. (See [67]). This formula is manipulated to allow for a user-supplied required error rate and current bitrate to be converted to an appropriate signal-to-noise ratio. (The formula used to determine turn-on and turn-off S/N ratios being $10 \log_{10} (0 - \frac{bitrate}{bandwidth} . ln (2.errorrate)))$.

The way in which most of these parameters affect the simulation will be obvious from the protocol diagram shown in Figure 59. The others are briefly discussed here:

- Turn on and turn off rates are respectively the error rates (which is converted to a signal-to-noise ratio by the system) at which it is believed the remote station will hear the probe and the rate at which shut-down occurs.
- Master and remote power determine the power alteration necessary on sample strength in order to correctly simulate the environment. (In the case of the measurement systems under study here, both figures are 400 watts, which involves no power alteration.)
- Path length is the determining factor for propagation delay.

3.8 Simulation Method

The main driver of the simulation is this Pascal routine :

```
BEGIN
    FIND_TURN_ON;
    WHILE ANOTHER_BLOCK_FEASIBLE DO BEGIN
        RECEIVE_PROBE;
        SWITCH_FROM_RECEIVE_WAIT;
        SEND_SYNC;
        WHILE ((NOT DIPPED_UNDER_MINIMUM) AND
                ANOTHER_BLOCK_FEASIBLE) DO BEGIN
            SEND_BLOCK( BLOCK_SIZE_XSECS, FRAME);
            WAIT_MAX_OF_2_TIMES_PROP_DELAY_OR
            SWITCH_FROM_TRANSMIT;
            SEND_BLOCK( ACK_SIZE_XSECS, ACK);
            SWITCH_FROM_RECEIVE_WAIT
        END;
        FIND_TURN_ON;
    END;
END;
```

The logic is as follows : The routine will go through trail samples in FIND_TURN_ON until a sample is found with a signal-to-noise ratio yielding an error rate below that of the specified turn-on error rate. At this stage the probe would be received, and the remote transmitter will switch from receive to transmit mode. The sync signal will then be sent to synchronise the two stations.

After this the remote transmitter will repeatedly send blocks to the central station. After each block sent there will be a delay of the greater of either twice the propagation delay between stations (for the block to reach the central station and an ACK frame to return), or the time it takes for the remote to switch from transmit to receive mode (to receive the ACK). The ACK is then received, and the remote switches back to transmit mode.

This looping process continues until either the ANOTHER_BLOCK_FEASIBLE test fails, indicating that the current trail does not have enough duration left to support the time another block would need on it, or the DIPPED_UNDER_MINIMUM test holds, indicating that the signal-to-noise ratio has fallen down to the turn-off error rate level. The whole process is then repeated until all trails for the period chosen (the results presented here use a period of one day) have been processed.

The SEND_BLOCK routine does not assume correct transmission of the block (this applies to the ACK frame as well as the data block) but rather determines the probability of the block's correct transmission, based on the signal-to-noise ratios during the time the block would be sent. The sum of these probabilities is the result given for blocks transmitted, as opposed to the sum of attempts to transmit blocks.

3.9 Search Technique

As might be apparent, the lack of variable block sizes in the real-world half-duplex simulation meant that merely finding an optimal bit rate, as was done with the preceding 'no-delay' results, was not possible. The optimising search had to be two-dimensional, with both block size and bit rate variable, to avoid making all results obtained specific to a particular block size.

An exhaustive search for optimality was clearly impossible -the simulation of thousands of possible sizes combined with thousands of possible rates would take years of computer time even on a supercomputer. A breadth and depth search was implemented instead, with the simulation checking rates and sizes at points equal intervals between user-specified maxima and minima. A breadth of four, defining the number of intervals, was used for the results presented here. The point at which the highest throughput was obtained in this search was then used as the midpoint for a new search, with minima and maxima now the earlier checked points on either side of the first-level optimum. This continued to a depth of three levels for the results presented here.

Obviously this method of search does assume some continuity on the 'landscape' where the plane is defined by bit rates and block sizes, and the height by throughput data bits. The search method is only guaranteed success if the landscape has exactly one peak, and all other points being either on slopes thereof or on the plane. If this is not the case then the technique is subject to the 'foothills' problem (see Winston [54]), where a sub-peak (local maximum) becomes the focus of the search, with the true maximum missed. A brief look at the full results at the final level of a search, presented in Appendix D, will show that a degree of discontinuity does in fact exist. However, the correspondence between the results presented here and the earlier

'no-delay' results does seem to indicate that this has not been a problem in these simulations.

3.10 Results of the Simulation

Figures 60 to 62 respectively show results for optimal bit rates found, optimal block sizes found, and throughput bits per second achievable using these optima. Each of these figures show the results for all trails, overdense trails only, and underdense trails only, for each of the three links. These results were based on simulations over 24-hour periods.

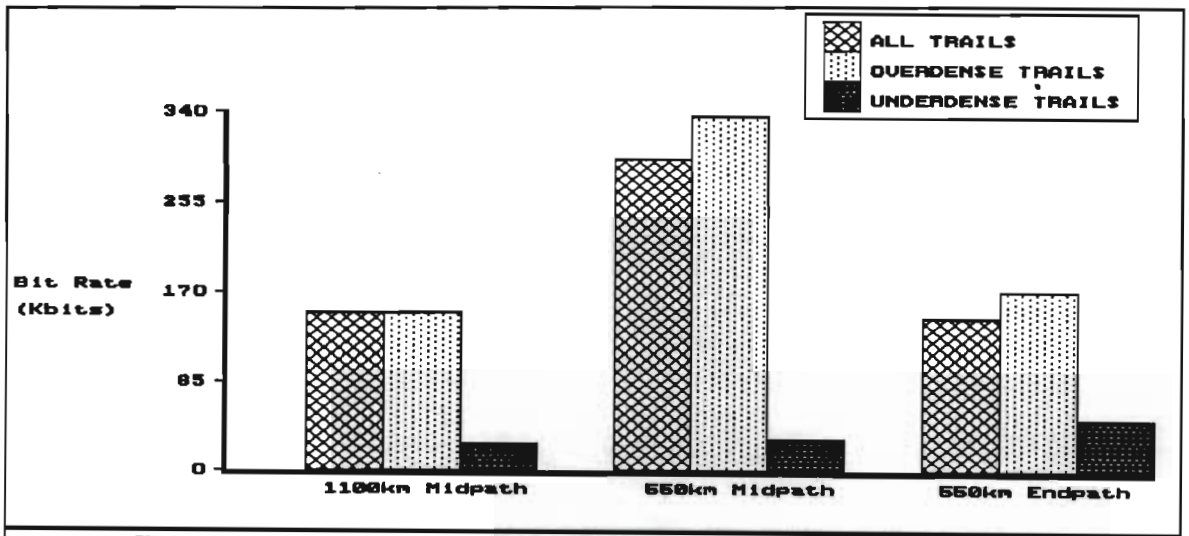


Figure 60 - Optimal Data Rates Found by Simulation

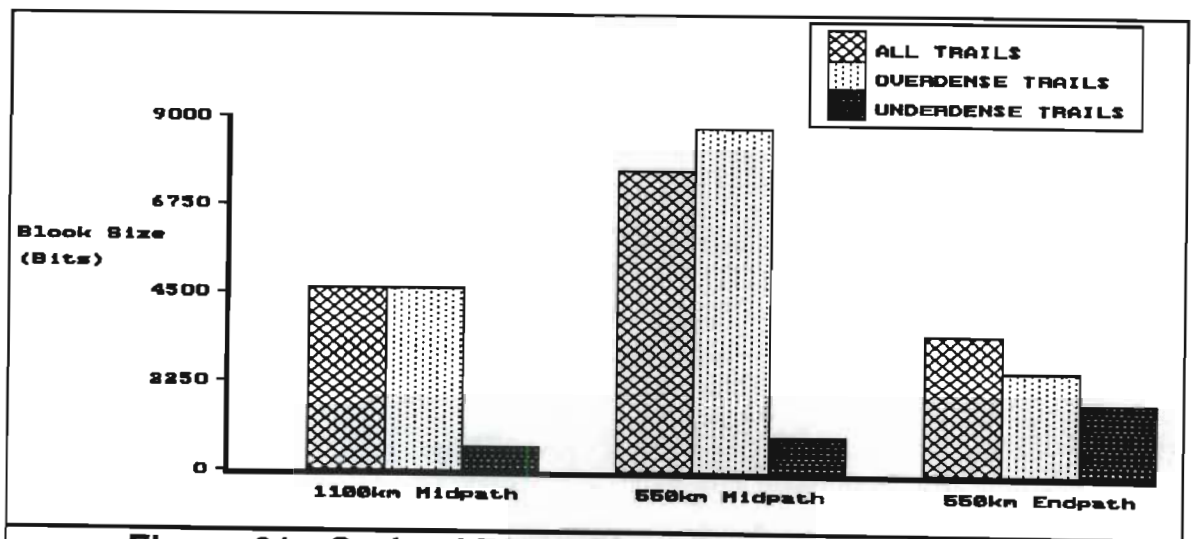


Figure 61 - Optimal Block Sizes Found by Simulation

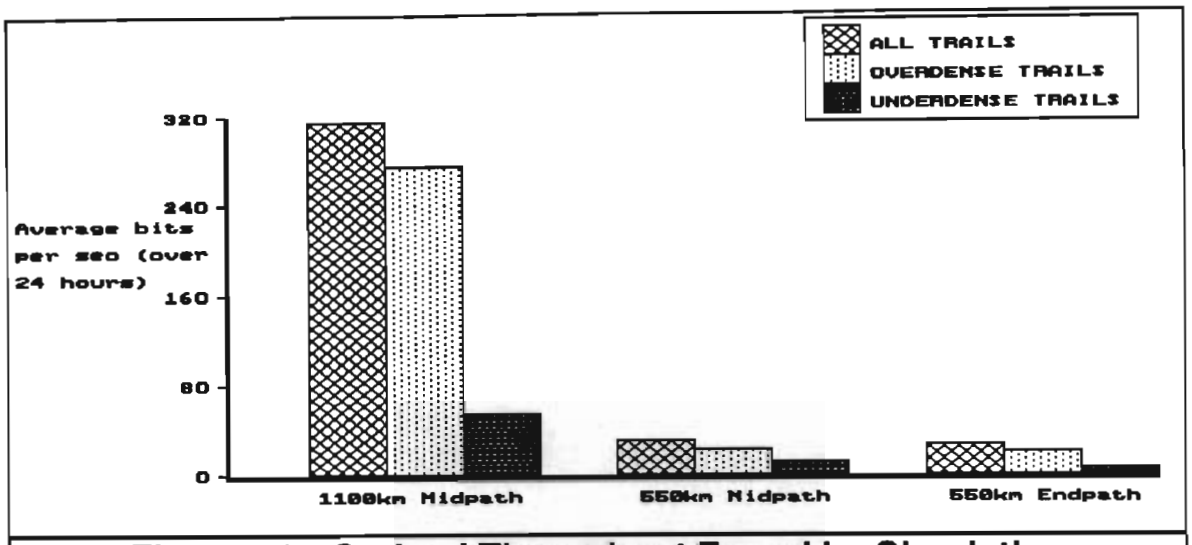


Figure 62 - Optimal Throughput Found by Simulation

As is apparent, the overdense group optima were always close to the overall optima, while the underdense group optima bear little or no relation to the overall optima. This gives strong support to the theory, implied by the 'no-delay' results, that the overdense group model should be the chief basis of determining system parameters such as bit rate in a throughput-oriented installation, with the underdense group used at most as only a minor modifying effect. (This is most valid at high or optimal rates, but still applies at low data rates, under five kbps or so, where the underdense group has a greater effect.) Importantly, the results found in this 'real-world' simulation are in accord with those found in the 'no-delays' simulation.

To further compare the relative importance of the different trail families, the half-duplex simulation was modified to store the results of which trail families carried the data sent. Figures 63 to 67 show the percentage contribution of the various trail groups to data communication at various bit rates and block sizes. These figures are based on simulations involving approximately ten thousand trails over the 1100 km link. Figure 68 shows individual trail family contributions at the optimal bit rate and block size for the link.

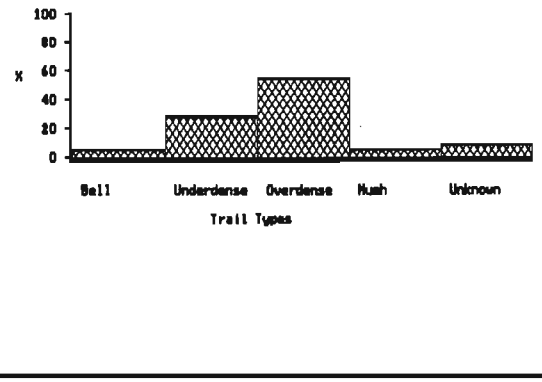


Figure 63 - Contributions at 5kbps, block size 200 bits

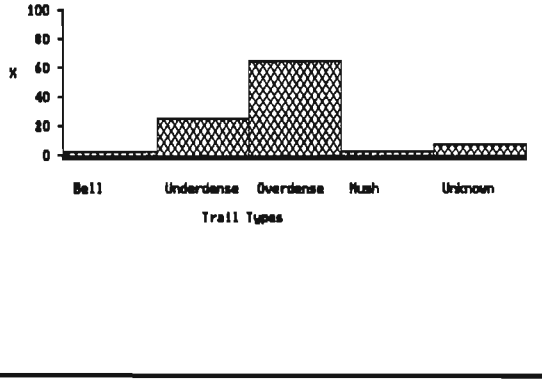


Figure 64 - Contributions at 10kbps, block size 400 bits

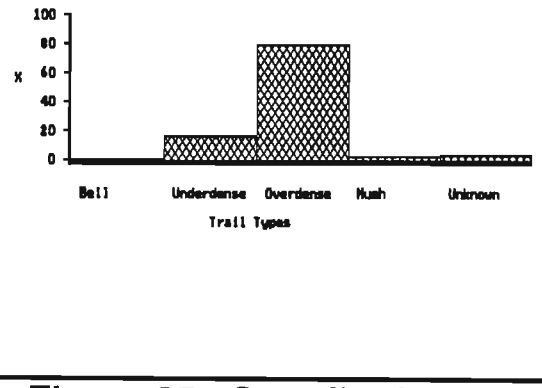


Figure 65 - Contributions at 50kbps, block size 2000 bits

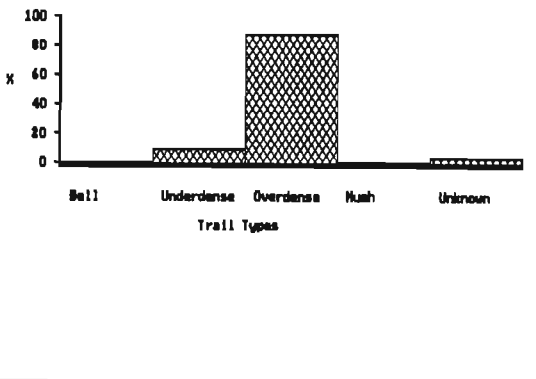


Figure 66 - Contributions at 150kbps, block size 4500 bits

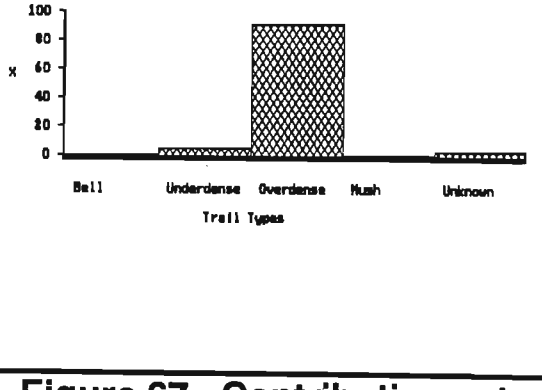


Figure 67 - Contributions at 250kbps, block size 7000 bits

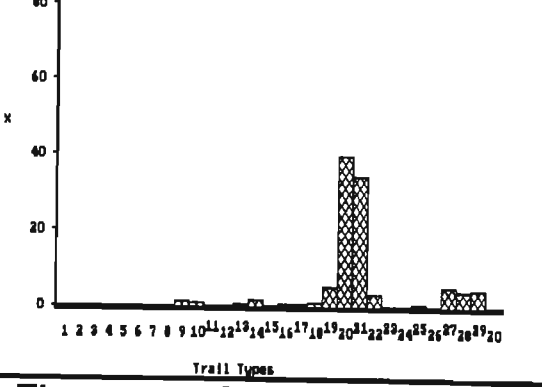


Figure 68 - Contributions at 10kbps, block size 400 bits

The dominance of the overdense contribution is obvious, and confirms the results presented earlier. This dominance is less pronounced at the lower rates and block sizes, where the larger counts of underdense trails have some significance, but the greater duration of the overdense trails still ensures that they are the major contributors even in these 'small' systems.

Figure 68 shows exactly which families in the overdense group were most useful for communications. Here the percentage contribution of each individual trail family, (as classified in [52]), is given for the simulation using 150 kbps bit rate and 4500 bits block size. This rate and size was chosen as these were found to be the optimal operating parameters, as shown earlier. The percentage contributions show that the types (families) 20, 21 and 27 are the major contributors, these being respectively the 'rectified sine' (with shape on time and amplitude axes being implied by the family name), 'non-sine' (clearly overdense trails which did not fit into any of the other five overdense families) and 'sinusoidal families' (again with the name indicating the shape on time and amplitude axes). This ties in fairly well with the observations on trail family contribution to duration discussed in Chapter 2.

Other non-negligible contributors are type 18, (the 'twins'), type 28, the 'wind-blown overdense' family, and type 29, the 'hump-backed classic underdense' family.

Figure 69 shows the adaptive rate results for different trail families. The upper line gives the average data rate achieved for all trails, the second gives the rate achieved when only using overdense trails, the third the rate achieved when only using the two major contributors within the overdense group ('rectified sine' family and 'non-sine' family), and the lower line the rate achieved when only using underdense trails.

Similar results to those shown in Figure 69 were obtained for the other two links. The dominance of the overdense group in terms of throughput contribution is once again confirmed by these adaptive data rate results, as is the importance of the 'rectified sine' and 'non-sine' families within this group.

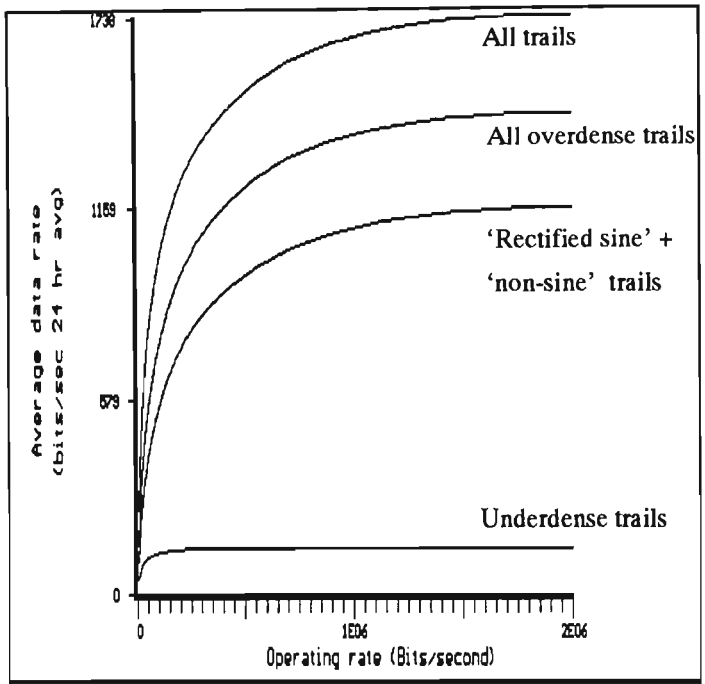


Figure 69 - Adaptive Data Rate Results by Trail Families

Figure 70 shows the normalised adaptive throughput across families. It is clear that the underdense results show little relation to the overall results, while the overdense results are virtually identical. Of particular interest is the fact that the 'rectified sine' and 'non-sine' results are very similar to the overall case. This suggests that these trail types could be used to model the adaptive gain of the meteor channel with a great deal more success than those models based on the underdense family.

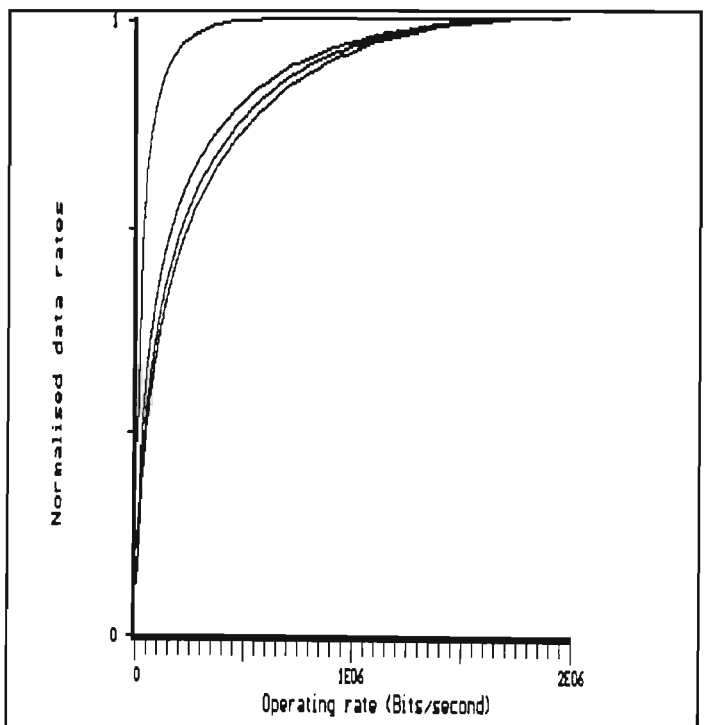


Figure 70 - Normalised Adaptive Gains by Trail Families

Upper line gives result for underdense trails, second gives result for all trails, third gives result for all overdense trails, lower gives result for 'rectified sine' and 'non-sine' trails.

3.11 Conclusion

The dominance of overdense trails, and particularly those of the 'rectified sine', 'non-sine' and 'sinusoidal' families, with regard to the throughput capacity of the meteor-burst channel is probably the most significant observation to arise from this work. It has been shown that in all the simulations - fixed rates, adaptive rates, and the half-duplex scenario - the overdense group carries the vast majority of data sent. This is true even at low fixed data rates such as 5 kbps, where the overdense trails still carry more data than the underdense trails. There is a decrease in the percentage of communications carried by the overdense group as bit-rate is reduced but they remain the dominating group at all rates.

In addition, across all simulations and all systems tested there is a high degree of correlation between the communications results on all trails and those on the overdense group. There is not such a correlation between all trails and the under-dense group. This suggests that the overdense group should be used as the basis for any general model of the meteor burst communications environment, with the underdense group being used only as a slight modifying factor.

A comparison between theoretical results as presented in [20, 49] and the measured results presented here has revealed significant differences. In both the fixed case, (see Figures 38 to 40), and the adaptive case, (see Figures 45 to 53), there is a marked divergence between the expected and actual results. This is particularly noticeable for the lower data rates, which has important implications as these rates are typically used in current working systems.

There does not seem to be any clear correlation amongst optimal fixed rates across different systems, or for different days on the same system. Thus a generally applicable 'best' fixed rate is not available. However, in all systems tested, there is a

definite peak of fixed rate throughput potential occurring strictly within the limits of 100 kbps to 2000 kbps. Thus the function involved was neither monotonically increasing nor monotonically decreasing in any of the simulations.

These results have been supported by the half-duplex simulations. These simulations have shown that there is a close correlation between the optimum data rates for maximum data throughput calculated using ideal fixed data rates, and those calculated using the half-duplex simulations.

The similarity between the form of the weighted rate density functions for the three systems (see Figures 45 to 53) shows that the functions are not highly dependent on meteor link characteristics. This finding should simplify the design of a practical adaptive data rate system.

The results presented have shown that using adaptive rates in preference to fixed rates results in significant improvements. The results of the fixed-adaptive simulation are of great interest and are fairly striking when one views Figures 55 to 57. There is a sharp rise in throughput as one increases the maximum possible modem rate at the lower range of data rates, but the gains fade sharply to rapidly diminishing returns as these increases continue. This has major implications for the design of adaptive bit-rate modems.

The 'modified overdense' model proposed by Weitzen seems to correlate fairly well with observed results as far as adaptive rate capacity of the channel is concerned, although the latter indicate a greater rate of increase in throughput as operating rate increases at low data rates.

The discrepancies between current theory and actual observed results indicate a need for further theoretical investigation in order to derive a model more compatible with actual meteor channel characteristics.

4 Wait Time in Meteor-Burst Communications

Abstract

Most studies of the communications potential of the channel have concentrated on channel throughput. The question of how much data the channel can carry is an essential concern, but so too is the question of how long it will take to send a message. This latter issue, wait time, has received considerably less attention, and is addressed by this chapter. Results of simulations using data recorded by practical monitoring systems are presented, allowing a comparison of wait times amongst various combinations of data-rates and message sizes. The meteor trail families contributing at these rates and sizes are identified.

Material in this chapter has been accepted for publication in the form of a paper entitled 'Wait Time in Meteor-Burst Communications', by SW Melville and JD Larsen, by the journal Transactions of the SAIEE.

4.1 Introduction

Research into meteor-burst communications (MBC) has largely concerned itself with throughput considerations (see [19, 20, 24, 55]), with wait time issues being largely ignored. However, for many of the real-world applications of MBC the delays involved for the transmission of a single message is of more concern than the total throughput capacity of the channel. While throughput and wait time considerations become equivalent when there are an infinite number of equal priority messages to send, this is clearly not the case when the delay in sending a single high-priority message is at issue.

In order to gain some insight into wait times, data captured on practical monitoring links were used. This chapter discusses the results of communications simulations using these data.

The simulations were made using three different message sizes, ten, eighty and two hundred and fifty-six bytes, these message sizes chosen to represent a typical command, a line of text, and a packet of data respectively. A range of data rates was employed.

The contributions made by the various groups or families of meteor trails, (eg. underdense family, overdense family) to message transmission at the various combinations of message size and bit rate is studied and results presented. The groups considered are 'overdense', 'underdense', 'bell', 'mush' and 'unknown'. These groups are supersets of the families identified in [52], as described in Chapter 3. The monitoring links used for this study were the 1100km and 550km midpath links described in Table 1 (see Chapter 2).

4.2 Method

Using Equation 1 (see Chapter 3), it was possible to determine the signal to noise ratio that would be required to allow the use of different data rates. Given a particular data rate, the simulation system can thus determine the signal to noise ratio necessary to maintain this rate, as well as the time interval that would be needed at that rate to send a given length message.[+]

+ In order to constrain the problem, actual transmission of messages was assumed to be purely a function of data rate and message size with other practical delays (eg. switching time, propagation delay) not considered.

Determining average wait time over a period then reduces to simply finding the number of trails reflections in the period which have signal to noise ratios at or above the required level for the required time. Dividing the period involved by this number gives average wait time. For this study periods of twenty-four hours were used in order to avoid possible skewing of results due to daily cyclical variations.

Note that this work is based on determining the average wait if there is **one** message to send, and so does not allow for multiple sends on the same trail. This work concentrates on determining wait time in the situation where there is a single high-priority message of size 10, 80 or 256 bytes, and does not consider situations where there are larger messages or multiple messages.

4.3 Results

Figures 71 and 72 below show average wait times for the two monitoring systems at different data rates. In each case the lower line on the graph is that of the 10-byte block, the middle line is the 80-byte block, and the upper the 256-byte block.

One would expect that the larger messages would have longer average waits than the shorter ones. The results show that this to be true at the lower data rates. Here many trails yield the comparatively low signal to noise ratio required, but most do not have the duration to allow transmission of the larger blocks. However at higher data rates the difference in delays experienced amongst message sizes is far less pronounced. In order to use the higher rates a large signal-to-noise ratio would be needed. Typically such ratios would only be found in high-amplitude overdense trails such as the 'rectified sine' family (see [51, 55]). As these high-amplitude trails are also long duration ones (see [56]), most trails which can support the high data rate at all will also be able to support it for the duration necessary to transmit the longer messages.

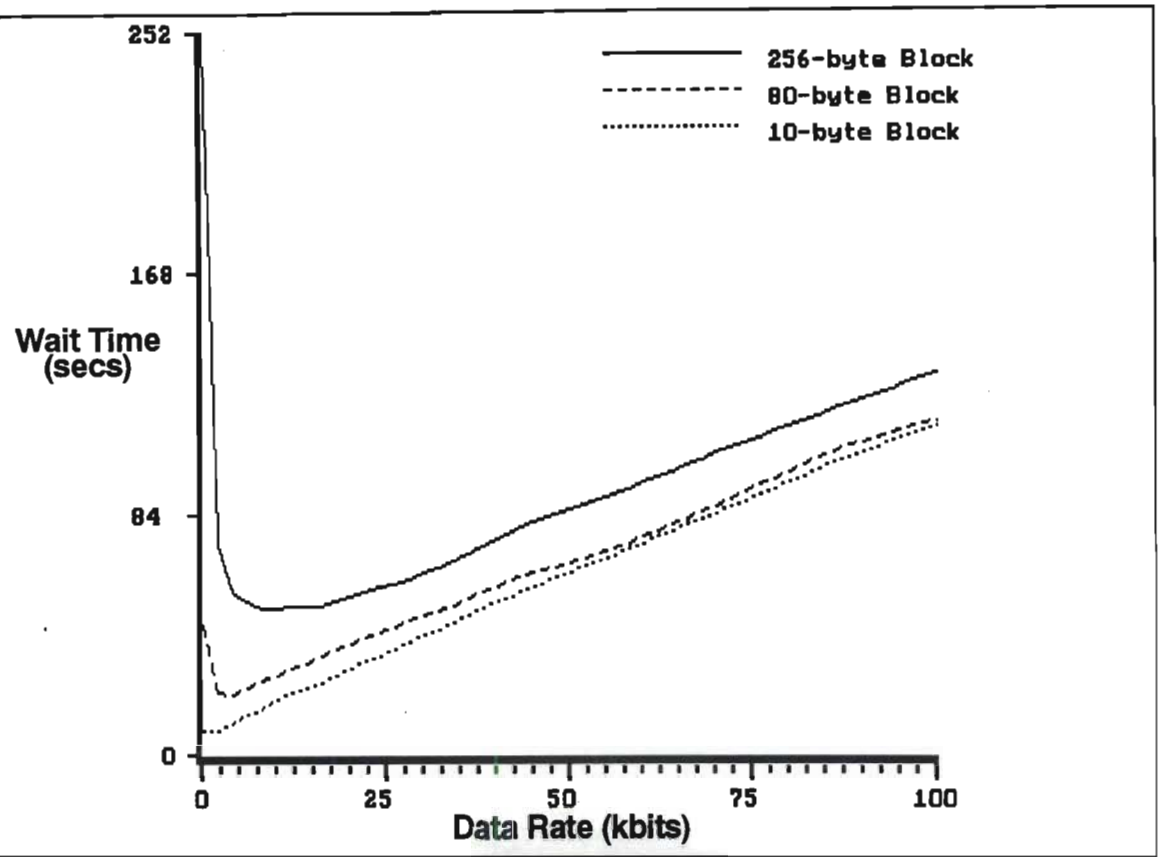


Figure 71 - Wait Time on 1100km Midpath Link

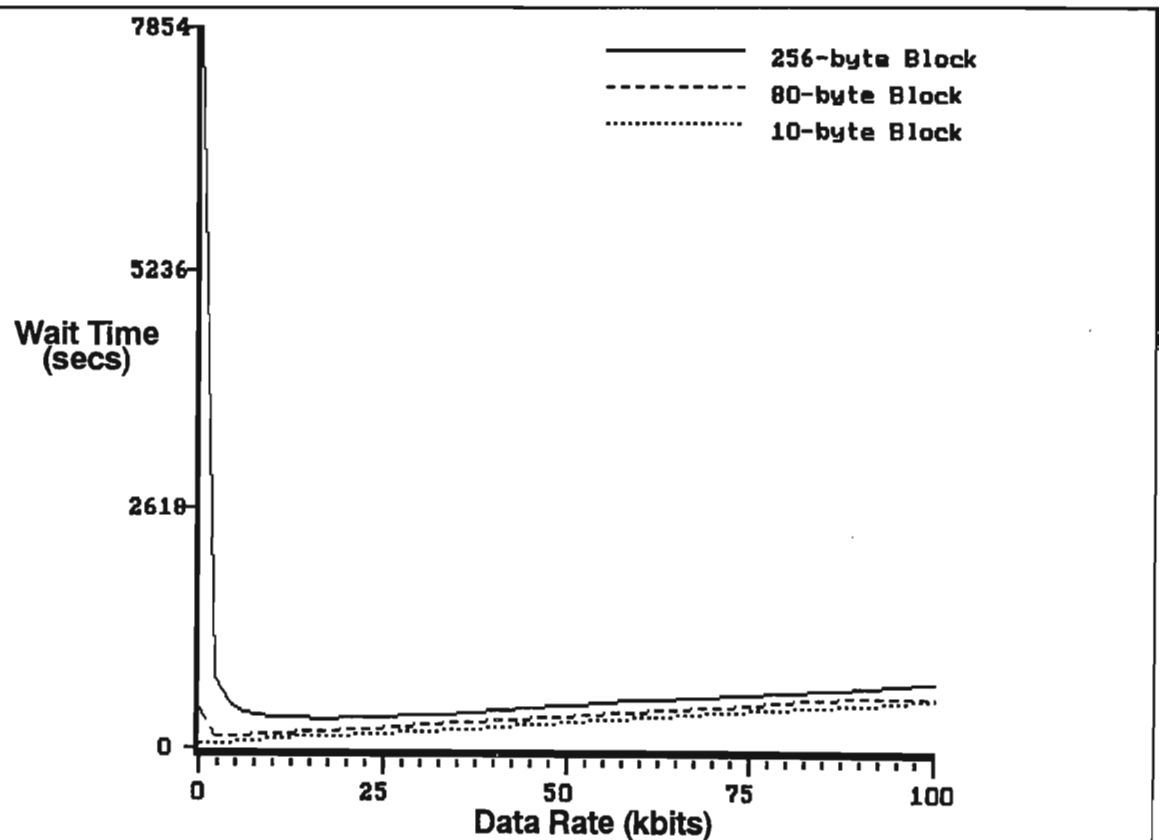


Figure 72 - Wait Time on 550km Midpath Link

This reasoning is supported by the fact that at rates of 450 Kbps (not shown in figures in order to allow greater resolution at the lower rates) the average waits for all three message sizes are identical.

The data rates at which wait time is minimised are of course of considerable interest. While considerably more trails were monitored on the 1100km link, and thus far smaller average waits were encountered, it is apparent that the rates at which wait time is minimised are much the same for both links. That such similar rates are found in systems not only monitoring different numbers of trails, but also receiving over different paths using different antenna configurations, is most interesting.

That these rates are all below 25 kbps is in sharp contrast to the optima for throughput capacity (over 100 kbps) found in [55]. As the same data was used for both that work and this, it is clear that different considerations must be employed according to whether a particular application requires optimisation to minimise wait time or to maximise throughput.

The reason for the different optimal rates would seem fairly obvious - in choosing a fixed data rate to maximise throughput the efficient use of the 'large' trails is the determining factor (see [55, 56]) while in minimising wait time the main concern is using as many trails as possible. Here data rates should be such that the far more numerous 'small' trails (underdense and 'bell' families), can be utilised.

The trail type (family) percentage contributions to message transmission at different combinations of message sizes and rates is shown in Figures 73 to 78. Results are from the 1100km link, (550km link results were much the same) using two different bit rates (4 kbps and 140 kbps) for each block size.

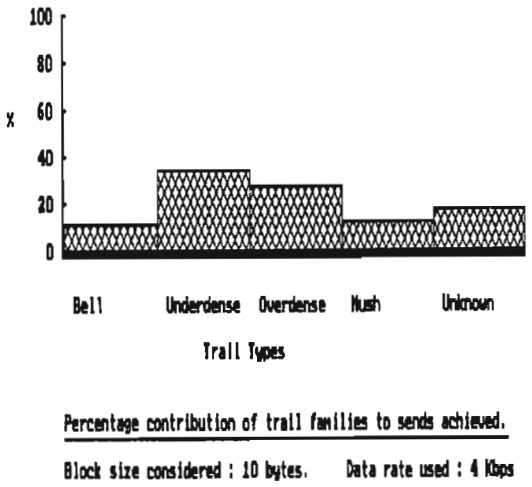


Figure 73 - Percentages with 10-byte block at 4 kbps

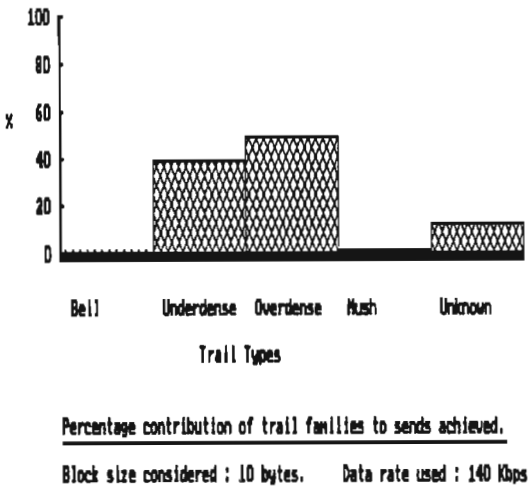


Figure 74 - Percentages with 10-byte block at 140 kbps

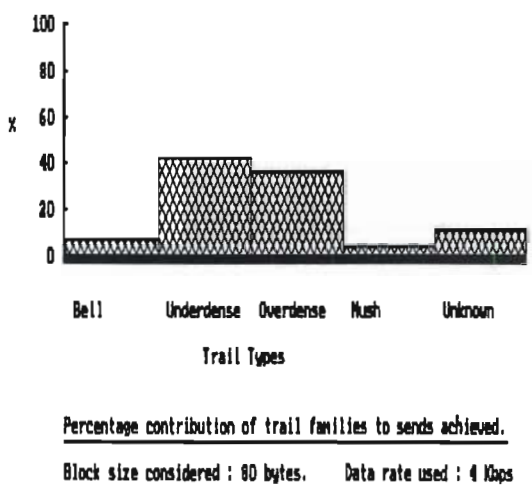


Figure 75 - Percentages with 80-byte block at 4 kbps

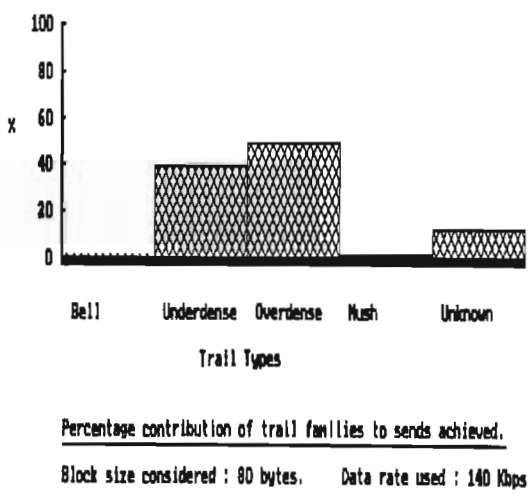


Figure 76 - Percentages with 80-byte block at 140 kbps

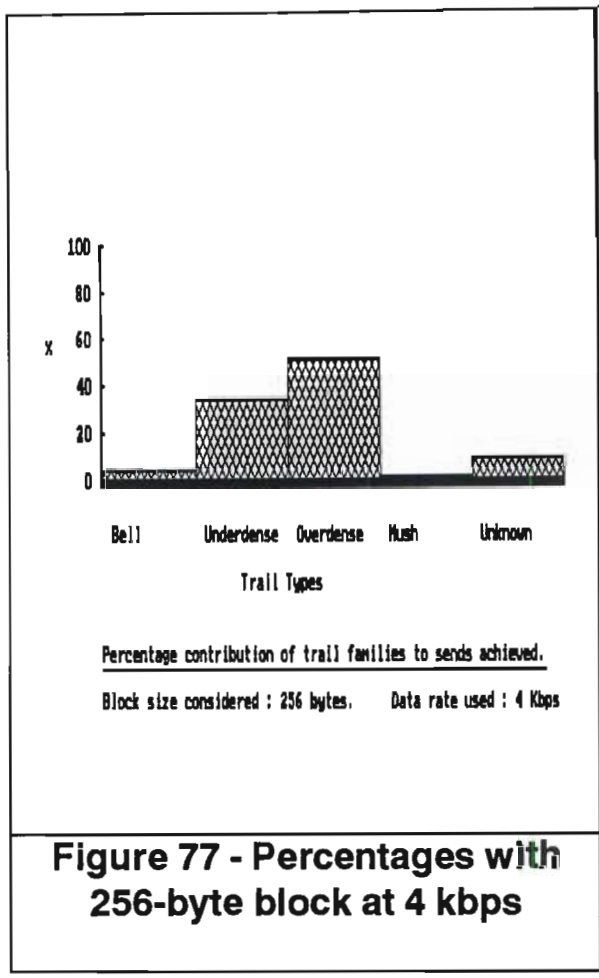


Figure 77 - Percentages with 256-byte block at 4 kbps

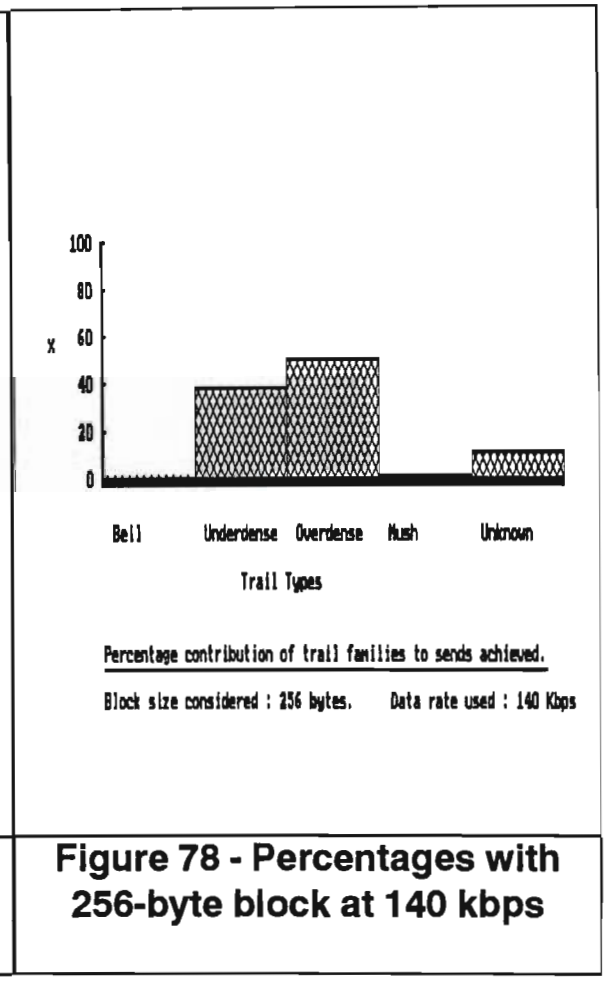


Figure 78 - Percentages with 256-byte block at 140 kbps

The figures show results that were, by and large, to be expected. The short duration, low signal strength mush and bell groups were only of any significance with low bit rates and small message sizes, and even in those situations were minor contributors. The unknown group had much the same contribution at all bit rates and message sizes, which is in line with the presumption that there is no significant pattern of duration or signal strength within the group (if there were such a pattern it would of course have shown the classification schema to have been inadequate).

As to the underdense/overdense contributions, the 10-byte and 80-byte results clearly show how the underdense will predominate at the lower rate while overdense trails will be of greater importance at higher rates. Figures 77 and 78 show a different picture with the 256-byte block however. While overdense trails are the major contributors at both rates (the bigger block size favouring these generally 'larger'

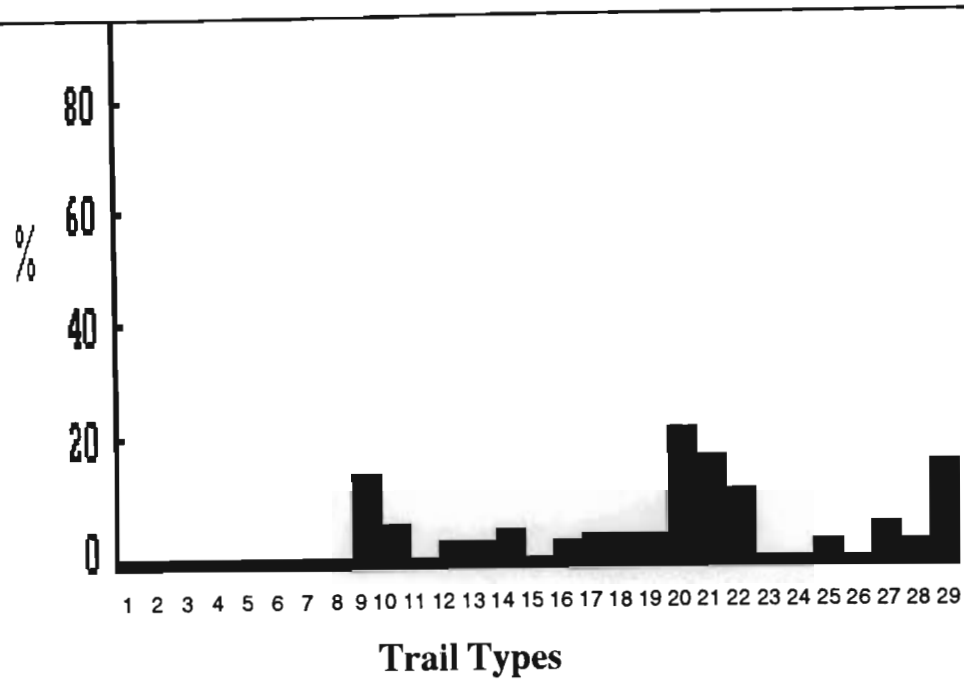
trails) it is noticeable that the underdense contribution actually increases at the higher data rate.

This somewhat surprising result can be explained by the fact that the sharp rises and relatively short duration of underdense trails ensures that many of these trails, while having the signal strength to sustain the high rate for the short transmit time needed at that rate, do not have the duration to be used for the longer time necessary at the lower rate.

This premise seems to be supported by the fact that at lower bit rates, such as 1 kbps, the underdense contribution was lower relative to the overdense than it was at 4 kbps. This applies to all three message sizes, but most noticeably for the larger two.

A study of very high data rates (500 Kbps) reveals that the overdense group is the sole contributor, for all message sizes, at these rates. This was to be expected as by far the greatest signal-to-noise ratios are found among members of this group. The fact that the largest contributor among the various overdense families at these rates was the 'rectified sine overdense' is in accord with the findings in [56] and [57].

The results presented have shown the contributions of groups which are supersets of the trail families defined in [52]. As an indication of the contributions of individual families, Figure 79 below shows percentage contributions at a bit rate of 56 kbps and message size of 80 bytes.



Percentage contribution of trail families to sends achieved.

Data Rate Employed : 56 kbps

Block Size Employed : 80 bytes

Figure 79 - All Trail Family Contributions

Type 20, the 'rectified sine overdense' is the major contributor, followed by types 29, 21 and 9 - respectively the 'hump-backed underdense', 'non-sine overdense' and 'classic underdense' families. Type 22, the 'gothic rocker' (unknown) family is the fifth largest contributor, while minor contributions are made by types 10 ('classic underdense with plateau') and 27 ('sinusoidal overdense').

4.4 Conclusion

A comparison of the results presented here with those obtained in work on throughput capacity presented in [20] and [55], clearly indicates that, for fixed data-rate environments, systems designed to minimise wait times should employ different data rates than those designed to maximise throughput.

It has been shown that different optimal rates are applicable for different message sizes, with the optimal rates becoming higher as the message size increases. The fact that the optimal rates for each of three different message sizes were duplicated in two different systems suggests that these optima might not be system dependent, although obviously a greater number of monitoring systems would need to be evaluated in order to ascertain the validity of this.

The fact that all optima found in both systems were at relatively low bit rates (under 25 kbps) indicates that current systems, which generally use such low rates, are particularly effective for minimising wait time. The trail family contributions presented in Figures 73 to 78 show that a very different set of trails is of interest in wait time minimisation than that used for throughput maximisation. In particular, the underdense group plays a far more significant role relative to the overdense group than in the throughput situation.

5 Networks in Meteor-Burst Communications I

Abstract

Communications networks with sporadically available links pose particular problems in terms of delay analysis. The SEER system (System for the Evaluation of Efficiency of Routings) was created in order to allow comparison and evaluation of the various strategies. The strategies considered included minimum-hop routing, shortest-path routing, and flood routing. The implementation of the simulator package for sporadic networks is discussed, with particular attention to a branch-and-bound algorithm devised for the efficient simulation of flood routing. The effects of network topologies and connectivity are analysed, and a number of results presented.

Material in this chapter has been previously published in the paper 'Analysing Routing Strategies in Sporadic Networks' by SW Melville, in Proceedings of the 6th South African Computer Symposium, July 1991. This paper is also to be published as an invited paper in the South African Computer Journal.

5.1 Introduction

In most forms of communication networks links between stations, once established, are constantly available. Thus the delay involved in sending a message from one station to another is essentially invariant if factors such as contention, buffer overflow and so forth are excluded from consideration.

Some networks, however, have links which are only sporadically available. Henceforth such networks are referred to as 'sporadic networks' in accordance with the terminology used in [58, 59]. Meteor-Burst Communications networks are clearly sporadic in nature. While billions of meteor trails are formed every day, a trail must

occur in the region of sky illuminated by both the transmit station and the receive station for the link between them to be viable. Different links will have different mean delays between usable meteor trails, due to such factors as transmitter and receiver power, galactic noise, size of common area of sky, and so on.

Modelling throughput over such sporadic links is not problematic (see [55],[57]) as the channel can merely be perceived as a low data-rate one rather than a collection of 'bursts' at high data rates. This is not valid in terms of delay analysis however (see [60]), as modelling the high data-rate sporadic channel as a low data- rate continuous one would result in gross underestimation of average delay for 'short' messages.

Some useful work has been done in analysing routing strategies for fast-changing networks (see [61]), and there has been a belief that this work might be applicable to sporadic networks ([62]). However there is a fundamental difference between these two types of networks. In the fast-changing network, typically one with mobile stations, the actual stations and established links change over time. In the sporadic network stations and the links between them are constant, but the links are only viable at certain times.

5.2 Routing Strategies Considered.

There is of course a plethora of potential routing strategies. In this work three of the more basic approaches are considered. These are minimum-hop routing, which seeks to route messages along the path containing the least number of intermediate stations; shortest-path routing, where messages are routed along the path estimated to have the least total delay; and flood routing, where the originating and all intermediate stations forward copies of messages to each of their neighbours which are not on the path the current copy originated from. In flood routing a record is kept

of what messages have been forwarded, so duplicate copies arriving later may be ignored.

If there was no variance associated with delay in a sporadic network, and no possibility of channel noise causing more than one attempt to be needed for each send, the time taken for a message using shortest-path routing would be identical to that taken using flooding (barring considerations like contention, buffer storage, etcetera), as the best flood path would of course be the shortest path. However in sporadic networks which do have a non-zero variance of delay between usable links, this is not always the case. While the path with the shortest **mean** delay can be established, there is no guarantee that this will be the shortest path **for a particular send**. Thus it is quite possible for the flood routing to find a path for a particular message which is 'shorter' (less delay) than the path of least mean delay.

5.3 Simplifying Assumptions.

A number of simplifying assumptions are made in order to constrain the problem of comparing different routing strategies. Firstly, it is assumed that there will be no contention within the network. While this may seem an extreme assumption in terms of conventional networks, it is somewhat more tenable for meteor-burst networks where the availability of any given link is independent of the availability of all others (bar the special case of reciprocal links). Certainly collisions will be infrequent in such networks, however there could still be some queuing contention. The circumstances in which this contention would arise, as well as its effects, shall be discussed in detail later in the chapter.[+]

+ In the section entitled 'Flooding and Contention'

Further, it is assumed that all stations have unlimited buffer storage, so no overflow problems can occur, that there is a constant transmission error rate across all links in the network, with a simple ARQ feedback scheme being employed, and that traffic flow between stations is uniform (ie. the network does not have any overly 'loquacious' or 'reticent' stations).

In addition, it is assumed that all messages are small enough to be sent within the time that a sporadic link is enabled - valid for command and control networks as well as most other MBC network applications (see [13, 35]) - and are not subdivided into packets.

Finally it is assumed that all delays (eg. propagation delays, switching delays) are insignificant in comparison to the delay experienced waiting for links to become 'enabled'. This latter assumption seems valid in terms of the meteor-burst environment, where delays waiting for links to become enabled typically are in at least the order of tens of seconds to minutes, while switching delays are in the order of milliseconds.

5.4 The SEER System.

A software package, System for the Evaluation of Efficiency of Routings (SEER) was designed and implemented in Pascal, using a clone of the IBM 286 AT. It accepts as input a file describing a network in terms of stations and links, with each link having two numbers associated with it. These numbers define the mean delay of link availability, as well as the standard deviation of this delay.

The system is presented with some number of message sends to be simulated, and randomly assigns sources and destinations for these messages. It then proceeds to simulate both the send of each message and the return of an acknowledgement of

its receipt, by each of the three routing strategies. Statistics are kept so that the overall mean delay and delay standard deviation for each strategy can be determined.

5.5 Sending a Message

Computation of the time needed to send a message and to receive an acknowledgement is achieved by a simple summing of the delays found for each link on each of the two routes being considered (the message going from source station to destination station and the ACKnowledgement returning from the destination station back to the source station).

Determining the delay for an attempted send over a particular link is achieved by using the following formula :

$$d_l = md_l + (r . \sqrt{3} . sd_l) \qquad \text{(Equation 2)}$$

where :

d_l is the delay for a send over link l
md_l is the mean delay associated with link l
sd_l is the standard deviation of delay associated with link l
and r is a random number in the range -1 to 1.

The decision to allow for a range of root three standard deviations on either side of the mean is based on this ensuring that the standard deviation of the generated delays will be the same as the standard deviation associated with the link. (The derivation of this result is given in Appendix E.) Note that in the case of d_l being negative (feasible if sd_l is large relative to md_l), d_l is set to zero. (To avoid the determination of a negative delay between available trails.) In such a situation results would obviously be skewed upwards, with the mean of the d_ls greater than md_l.

An actual send is determined to have occurred if a randomly generated number in the range zero to one is greater than the error rate specified by the user. If the random

number is less than or equal to this rate, the attempt fails. The time taken for the send across a link is then taken to be the sum of the delays for each attempted send plus the delay for the actual send.+

5.6 Route Determination

Determining shortest-path and minimum-hop routes is simple enough using Dijkstra's algorithm ([63]) for shortest paths. (For the minimum-hop case edge weights of one are given to all links, for shortest-path the mean delay associated with each link serves as its edge weight.)

Determining flood routing is a far more complex problem. Clearly determining delays along all possible routes in a network would be impractical for all but the smallest networks. (Consider the number of different routes between any given source and destination that would be encountered in a hundred node fully-connected network, for example.) However if the simulation is to be valid it is necessary to consider all routes which have the potential to yield the least delay in sending the message.

This problem was resolved by using what is essentially a branch-and-bound search as described by Lawler and Wood [64], and making use of what Ibaraki [65] refers to as 'pruning by dominance'.

The algorithm to send a message (or acknowledgement) from source to destination is described on the following page.

+ Note : the formula for delay given above is recomputed for each attempt, in order to correctly model the sporadic environment.

5.7 Flooding Algorithm

```
Set shortest completed path time to infinity
Create an incomplete path from source to destination consisting of just the source
node. Set 'time taken to here' for this path to be zero
Make this path the first (and only) element in a queue of paths
While the path queue is not empty do
    Set current path to be the front path in the queue
    Remove the current path from the queue
    For each neighbour of the last station on the current path do
        If the neighbour is already on the current path, ignore it (cycle)
        Else
            Determine the delay over the link from last station to the neighbour
            Set NewTime to the current path's 'time to here' plus this delay
            If NewTime is greater than shortest completed path time then do nothing
                further (not a candidate for quickest flood route)
            Else
                If the neighbour is the destination station then
                    Set shortest completed path time to NewTime
                Else
                    If the neighbour is on no other path then
                        Make a duplicate of current path, and add the neighbour
                        to the path. Set the path's 'time to here' to NewTime, and
                        place it on the queue
                    Else
                        Compare Newtime to the time taken to 'time to here' at the
                        point where the neighbour was reached on the other path.
                        If NewTime is greater then do nothing further (duplicate copy
                        being received, so can ignore path), else delete the other
                        path from the queue, and construct and add to the queue
                        the path created by adding the neighbour to current path
                        (in this case the other path would have carried the duplicate
                        rather than the first copy received, and so would not be
                        pursued)
    End (of 'for each neighbour' loop)
    Sort queue by 'time to here' of paths, with least-cost paths at front of the queue
End (Of 'while queue not empty' loop)
```

This algorithm guarantees finding the shortest route of the flood, as will be explained in the following paragraphs. In addition, it prunes the search space in such a way as to allow computation time to meet reasonable constraints.[+]

By considering all incomplete paths until they become longer (greater delay) than some completed path, or until some station on the path is found to have been reached earlier by another route (in which case the station would have ignored the 'duplicate copy' of the message and so the path would not have been continued on to the destination station), we ensure that no candidate for the shortest flood path is ignored.

At the same time, by sorting the queue of incomplete paths by time taken (path cost) to date, we ensure that the most promising alternates are explored first. This will tend to allow 'expensive' paths to be safely excluded as candidates early in the search. (These paths will be excluded when some other candidate yields a quicker route to a station on the path, at which stage we can safely prune by dominance as a path with a shorter route to a common intermediary of necessity cannot yield a longer route to the eventual destination; or when the cost of the incomplete path exceeds that of the shortest complete path.)

The structures used to implement the queue and paths are pointer records. The queue records allow for an ordering of the paths (each queue entry points to some incomplete path as well as to its successor in the queue), as well as holding fields to allow easy sorting and comparison on the basis of time of paths and stations involved, while the path records simply keep times to each station (to allow comparison of time

+ Greater detail on the degree to which pruning is achieved is given in the section entitled 'A Note on Speed' later in this chapter.

taken to intermediaries amongst paths) as well as keeping the correct sequence of stations in the paths.

The pointer structures used for queues and paths are shown below :

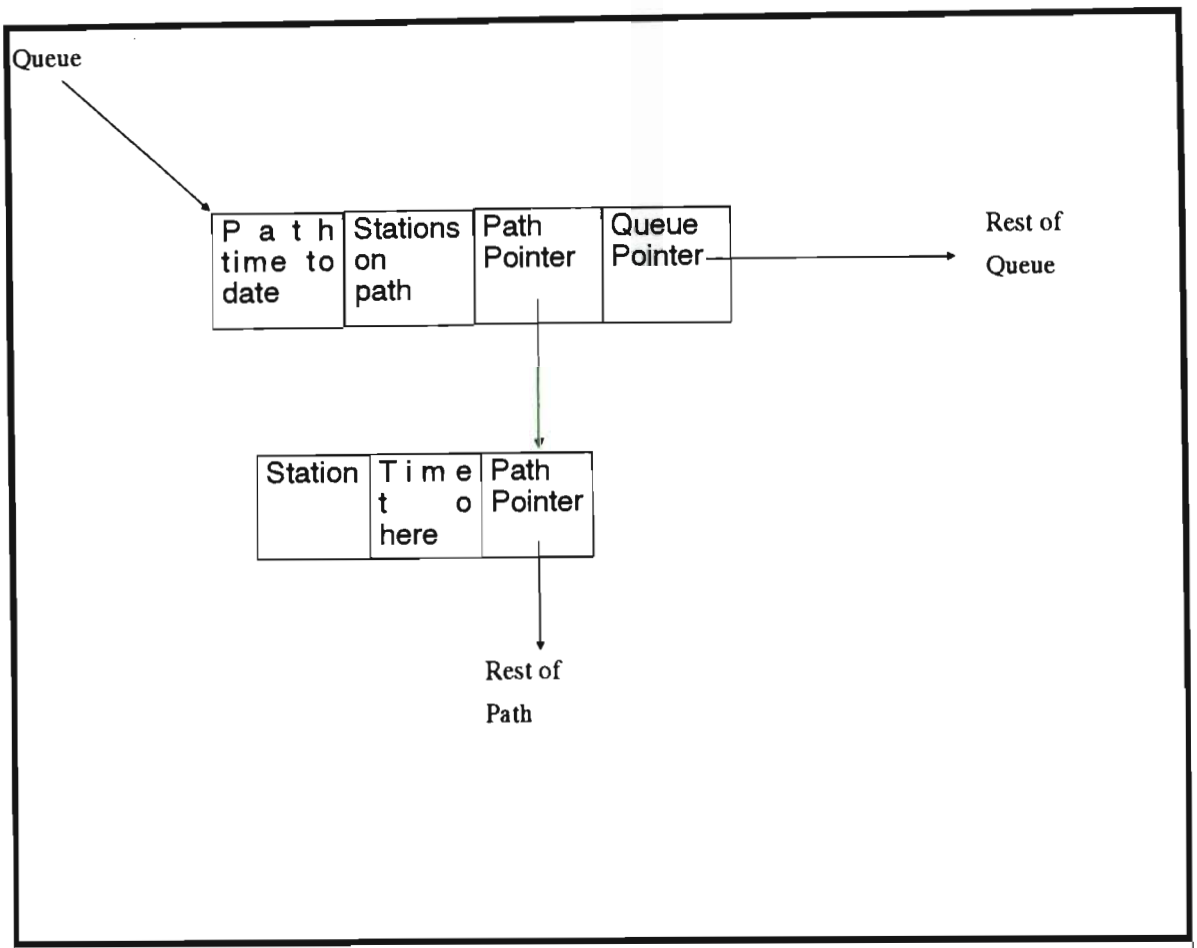


Figure 80 - Pointer Structure used For Flood Routing

5.8 Results

Results are given both for a 6-station network with varying mean delays and delay standard deviations, with delay time much as might be found in Meteor-Burst networks and for a fully-connected 50-station network with all links having identical delay characteristics.

A 'Meteor-model' network is shown below. The numbers on links indicate the mean delay and, in brackets, the standard deviation of delay, in seconds, for each link. Note that the network has been constructed to have reciprocal links, so that only one pair of numbers is given on each connection between each pair of stations, despite there being two links between them.

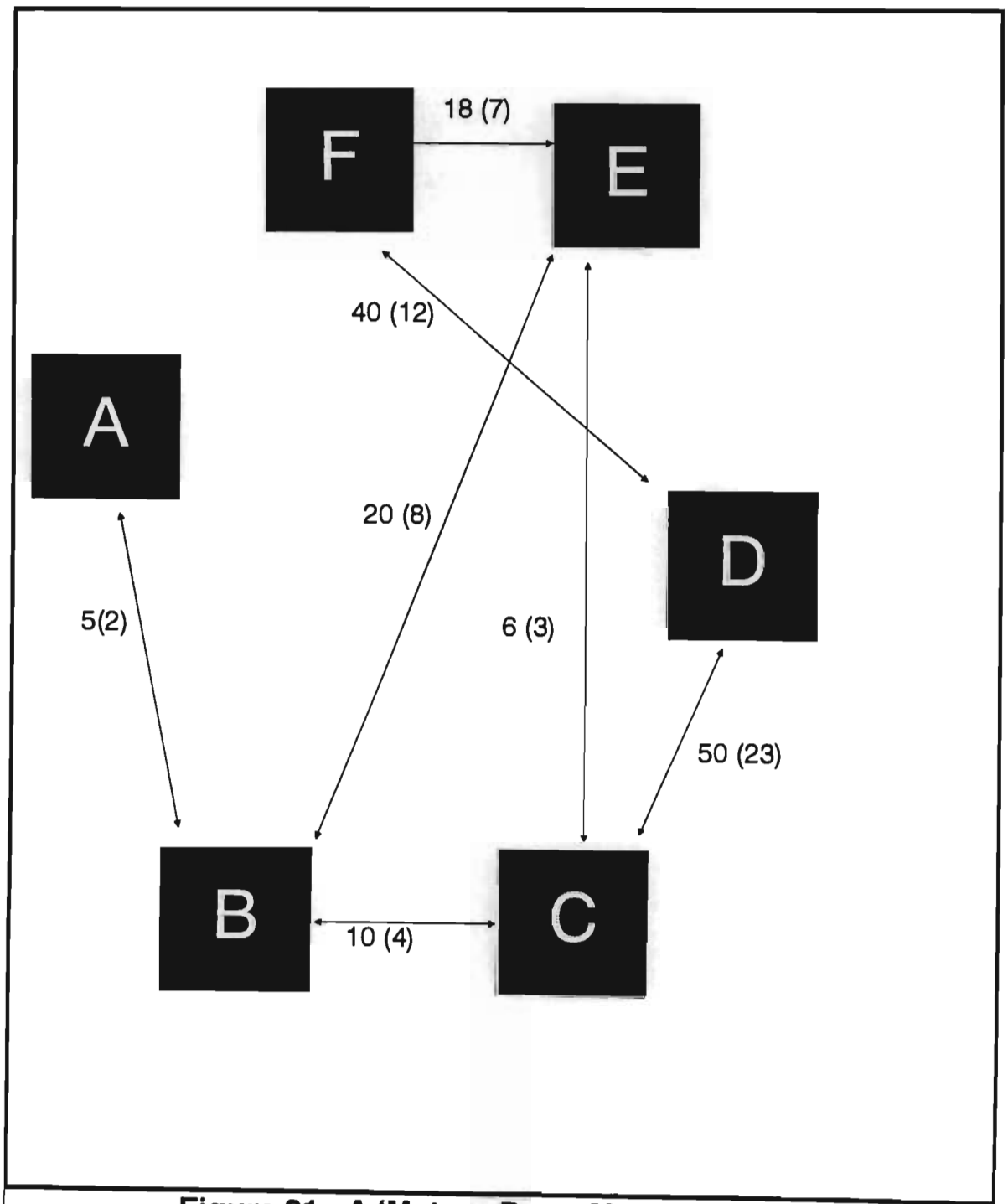


Figure 81 - A 'Meteor-Burst Model' Network

Results obtained by the SEER system simulating the transmission of one thousand messages on this network at various message error rates are shown in Figures 82 and 83 below (the full numeric results appear in Appendix F) :

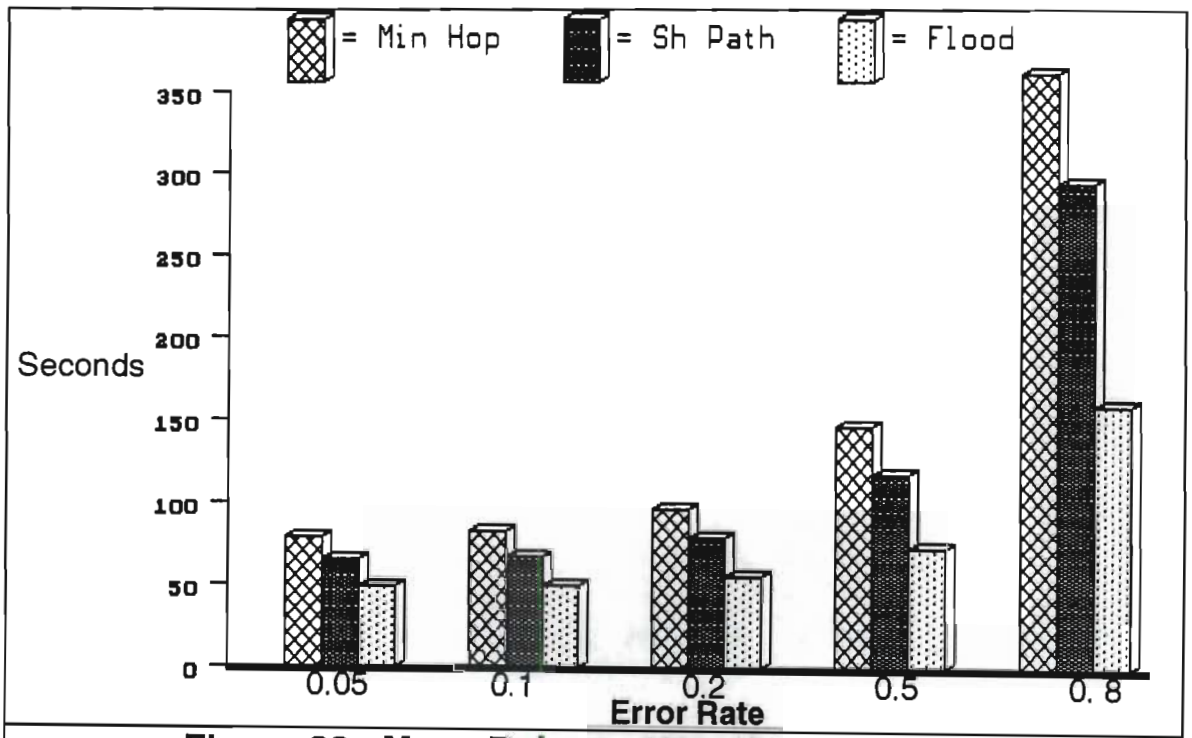


Figure 82 - Mean Delays of Routing Strategies

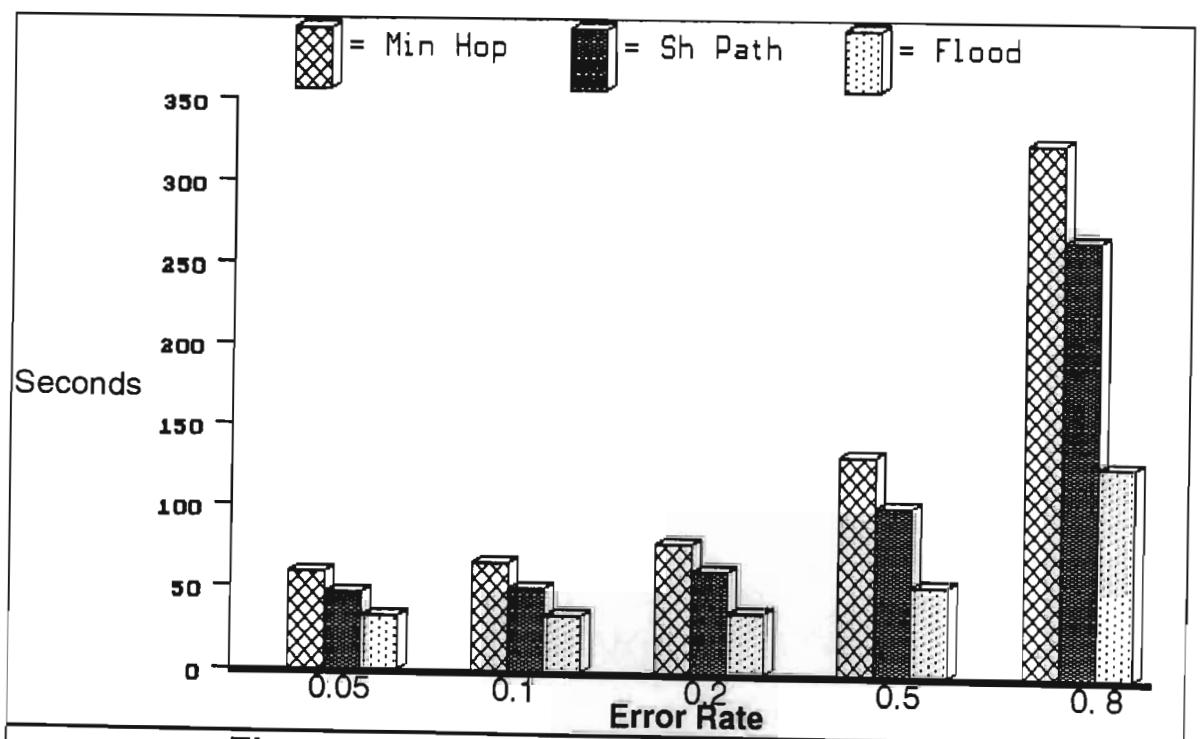


Figure 83 - Standard Deviations of Delays

Considering the lower error rates first, it is obvious that shortest-path outperforms minimum-hop. (In this network not all least mean delay paths are also least intermediary paths, obviously if this were the case the two strategies would perform equally well.) Flooding is significantly better than shortest- path, having a mean delay of only about 75% of that of shortest- path, and a standard deviation of delay of about 70% of that of shortest path, for the 0.05 error rate. At this low error rate the gains achieved by flooding can probably be attributed to its finding shorter delay paths for particular sends than achieved by using paths with the shortest mean delays.

Now what is most interesting is how robust the flood routing is in the face of higher error rates. At the 0.2 error rate we see that flood now has a mean delay of around 69% of that of shortest-path routing, and a standard deviation of only about 60% of that of shortest-path routing. This relative gain increases as higher error rates are encountered, as can be seen in Figures 84 and 85 below.

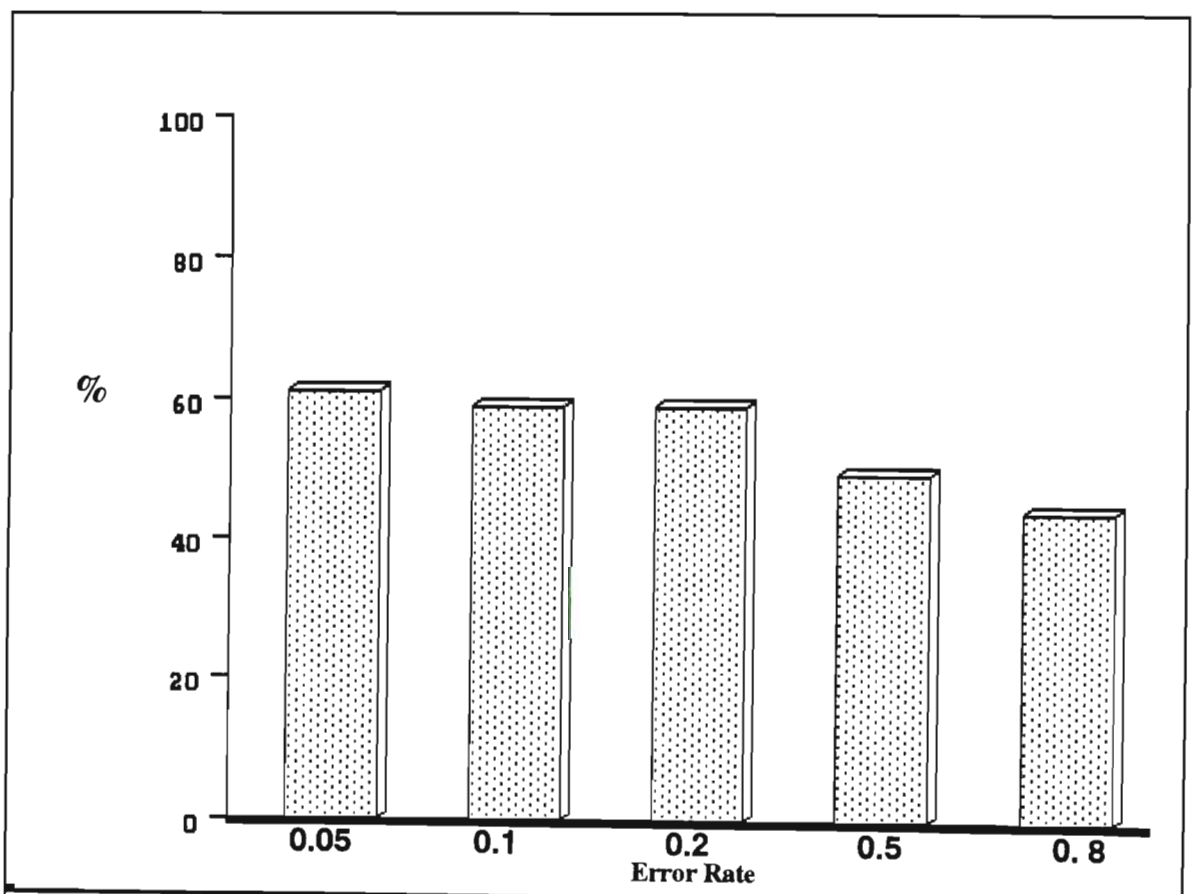


Figure 84 - Delay : Flood as a Percentage of Minimum-Hop

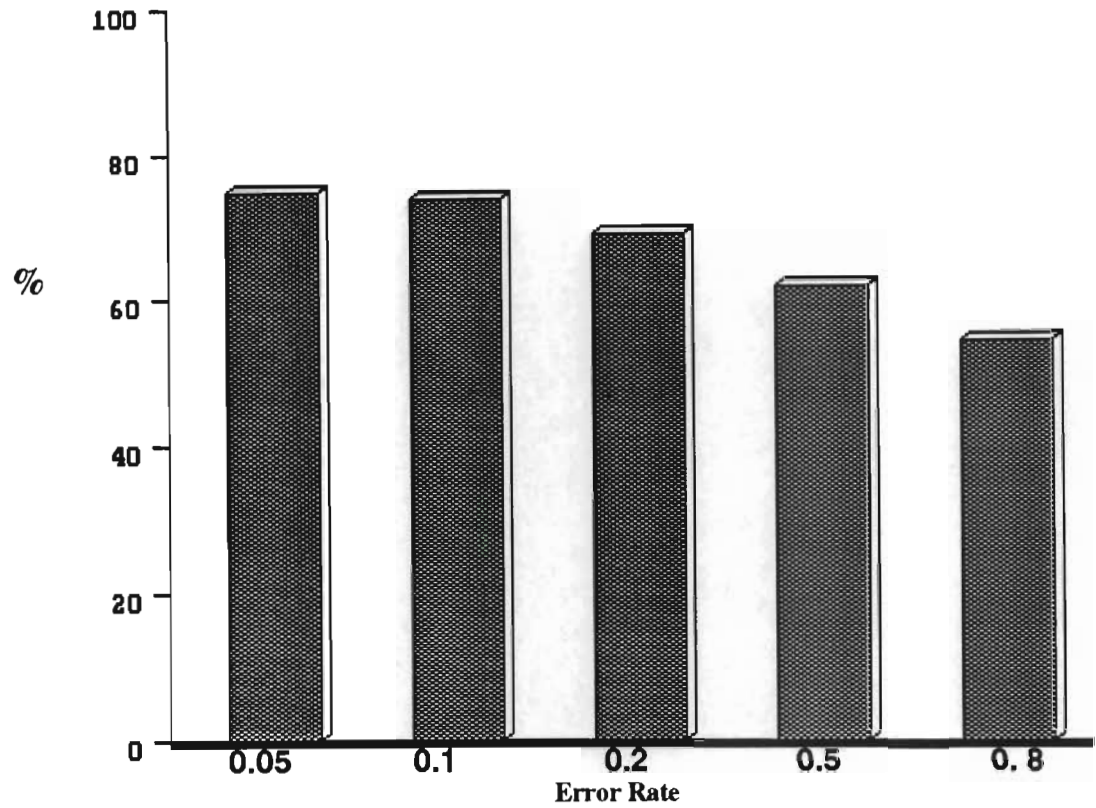


Figure 85 - Delay : Flood as a Percentage of Shortest-Path

This evidence that flood routing becomes relatively more attractive the noisier a channel gets can be attributed to the fact that in shortest-path or minimum-hop routing a failed attempt to send will generally result in considerable delay, while in flooding the failure will not have a great effect on delay if the link can be 'bypassed' by an alternate route taking the same time, or only a little more.

This 'bypass potential' also has considerable effect on the standard deviation of delay, as can be seen from the results. This is especially important to situations where 'worst-case behaviour' of a sporadic network is an important concern, such as would be the case in military networks.

The second network tested was a fully connected 50-station network, with each link being assigned a mean delay of 12 seconds and a delay standard deviation of 7

seconds. The simulation was run on 200 messages for a total of 400 messages and acknowledgements being sent.

Results for fully-connected network :

Error rate = **0.05**

Routing strategy	Mean Delay	Delay Std Deviation
Minimum-Hop	24.95	11.63
Shortest-Path	25.61	10.63
Flooding	3.47	1.26

Shortest-path routing and minimum-hop routing are equivalent in this case, with both strategies utilising the direct links between sources and destinations (the minor difference in results being due to the random factors in the computation).

Flood routing gives remarkably better results than the other strategies on this network, with less than 14% of the mean delay and less than 12% of the standard deviation encountered with the other strategies. The mean delay does seem exceptionally low at first glance, however on considering the topology of the network it becomes obvious that such a result should be expected. The range in which generated delays can fall is from zero seconds to a fraction over 24 seconds. This means that we would only expect the delay on the direct link between source and destination to be 'low' (say two seconds or less) about $^1/_{12}$ of the time. Two-link paths will have delays in the range 0 seconds to 48 seconds, and would thus have delays of two seconds or less about $^1/_{24}$ of the time. However, there are 48 two-link paths in this fully-connected 50-node network, so we would expect two of these ($48 * ^1/_{24}$) to have such delays. Thus without even considering the other potential paths (3-link, 4-link, etcetera), it is clear that such low delays for flood routing in a highly-connected topology are not only possible but should in fact be expected.

5.9 A Note on Speed

Clearly the most time-consuming section of code in the SEER system is that dealing with the generate-and-test involved in the flooding simulation. The performance of the algorithm used shall be briefly discussed.

The algorithm has a best case scenario occurring when there is a direct link between source and sink, and this link has the least delay amongst all the links emanating from the source. Here the algorithm will detect the optimal flood path after inspecting only the links connecting the source to its neighbours, which gives a gain over exhaustive search equal to E_s/E_n, where E_s is the out-degree of the source, and E_n is the number of edges on all source-sink paths.

The worst-case scenario would occur in the unlikely situation where all paths between source and sink are node-disjoint (no common intermediaries would prevent any pruning by dominance) **and** all path delays from source to the station immediately preceding the sink are greater than the shortest complete path from source to sink. This latter condition means that no partial path could be discarded until its entire delay was computed. In this worst-case scenario the search algorithm would give no gain whatsoever over exhaustive search.

Determining an average case would be a particularly difficult exercise, if indeed it is possible to do so. Clearly the topology of the network and the pattern of delays on links between source and sink (do 'bad' paths become distinctly bad early?) would have to be considered in any such determination. However the algorithm presented here has constrained the problem well for the networks tested - for 400 floods on the fully-connected 50-station network discussed the simulation ran to completion in under an hour, while the simulations on the smaller network were handled in seconds. This on a 286 PC/AT clone.

5.10 Flooding and Contention

Queuing contention arises when a station has a number of messages to send, and the order in which they are sent will affect the delay associated with their transmission. Such contention can arise under any of the routing strategies - one needs only consider the hub of a star topology network to see this. However it is clear that the flood mechanism will be most prone to create such contention, and so we will consider only this case in determining at what point such contention would degrade network performance.

An important point with regard to sporadic networks is that queuing contention is only significant if the time period that the link is enabled between some transmitter T and receiver R is too short to allow **all** messages T has for R to be sent. (Given the earlier assumption that the delay between link availability is our only significant delay, it follows that there is no significant delay between a message transmitted immediately the link becomes viable, and one transmitted immediately before it is no longer viable.) In the worst case scenario of flooding, T would have a copy of every message in the network waiting to be sent to R when the link becomes enabled, and so contention would arise if the link was not enabled for a sufficient time to allow all these messages to be sent.

From this, we can see that a number of factors affect whether contention arises in a sporadic network or not, (and how severely it becomes a problem). Firstly, the amount of data in the network as a whole is a concern. The network can avoid contention if it is loaded to any level up to any point where the throughput possible on a link is greater than or equal to the amount of data being transmitted in the network. As loading becomes heavier than this so contention could arise, and the greater the loading the higher the risk, and the more severe the effect, of queuing contention.

Conversely, the greater the throughput potential of enabled links, (the amount of time they are enabled for, times the data-rate they can support), the higher the loading that can be achieved before contention arises.

Finally, of course, the topology of the network and the pattern of link availability will determine how severely loaded individual stations become. (ie. how bad our worst-case for a particular network can get.)

As an example in the case of Meteor-Burst Communications, a usable trail would be roughly 250 ms, and could support a data rate of about 32 kbps. This means that an enabled link would be capable of sending around 8 kilobits of data. For application networks with an average message length of 40 bytes or so this would be about 25 messages. Thus the network could be at risk of queuing contention if more than 25 messages were 'in transit' at any given time.

Determining the probability of such contention actually occurring when the network is loaded beyond this 'safe' level, and to what degree it would affect the efficiency of routing strategies, is a subject that needs further work. What can be seen, though, is that the effect of contention will be largely determined by the application area of the network. For current MBC applications such as command and control, where relatively short messages are the norm, contention will be unlikely to create any serious problems unless a very high number of messages are being sent. On the other hand, in an application where large volumes of data are being sent - executable files, for example - across the network, or where the normal loading of the network is high (many messages) then contention could well begin to create serious problems.

5.11 Conclusion

The design of the SEER system, in particular in terms of its handling of the problem of sporadic link availability and the branch-and-bound algorithm used to simulate flooding, has allowed for effective modelling and fast simulation of routing in a sporadic network environment.

Flood routing has clearly yielded significantly better results for sporadic networks than the other two alternates considered, in terms of both lower mean delay in the network as well as far lower standard deviation of delay. It becomes even more attractive in situations where transmission error rates are high. Shortest-path routing will have significant benefits over minimum-hop routing in some networks, but will obviously be equivalent in networks where the minimum-hop route and the shortest-path route are the same.

There are of course other factors to be considered in contrasting the routing strategies - overheads such as the construction of routing tables, queuing contention, problems such as buffer overflow and so on. However it does seem that the particular nature of sporadic networks will still make flood routing the most effective routing strategy for these networks.

These preliminary results indicate that the topology of sporadic networks, and particularly the degree of connectivity in networks, will be a major factor in expected delays. Chapter 6 will deal with these issues.

6 Networks in Meteor-Burst Communications II

Abstract

Initial results presented in Chapter 5 suggested that flooding was the optimal routing strategy for MBC networks, and that the connectivity and topology of MBC networks would be a major factor in delays experienced. Further work on analysis of delay with respect to network topology was done, and is presented here. Results indicate that delay on networks will decrease exponentially as connectivity increases linearly.

Material in this chapter has been previously published in the paper 'Networking Meteor-Burst Communications', by SW Melville and DI Carson, in Proceedings of the 4th South African IEEE Conference on Communications and Signal Processing, August 1991.

6.1 Introduction

Most current MBC networks are essentially star configurations based on a single master stations and several remote stations, [13, 14, 25, 35]. The inherent limitations of such topologies has led to a consideration of alternate configurations, and their associated delays.

In order to test wait time behaviour over various topologies in an equitable manner, all networks generated were given links with the same mean delay (20 seconds) and standard deviation of delay (14 seconds), and all networks contained exactly forty stations. Shortest-path and minimum-hop routing are thus identical, as the path of least mean delay is also the path with least intermediary stations. Error rate was set to a constant 0.05 in all cases, and simulations were run over a thousand messages

on each network. Topologies studied included trees, stars, and networks with varying degrees of connectivity.

6.2 Networks Considered

Three tree-structured networks with different branching factors were generated, in order to allow the study of the effect of branching factor on network delay. These networks are shown in Figures 86 to 88.

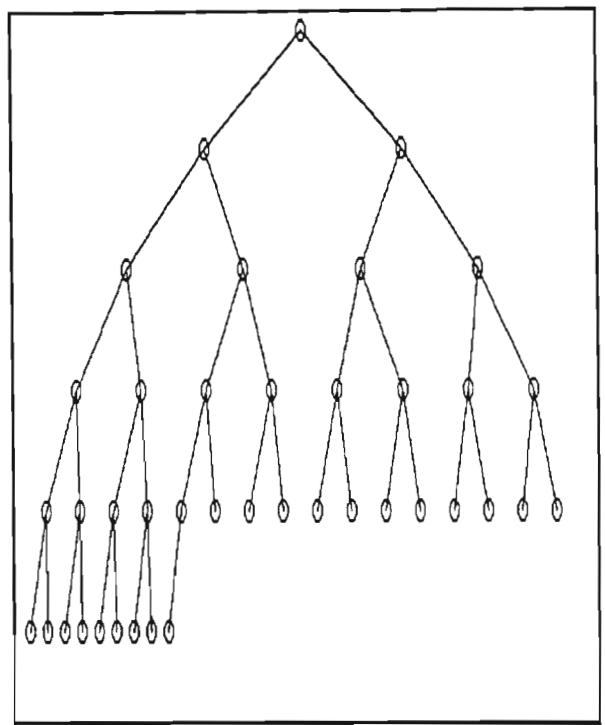


Figure 86 - Tree with Branching Factor of 2

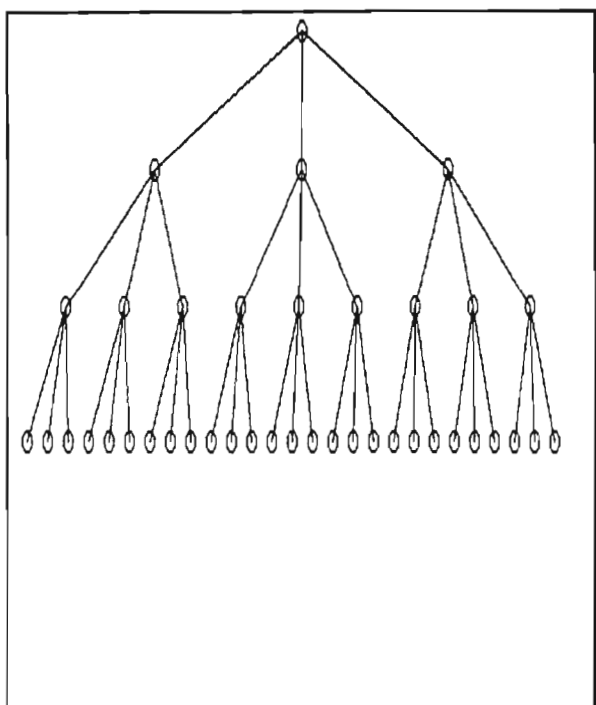


Figure 87 - Tree with Branching Factor of 3

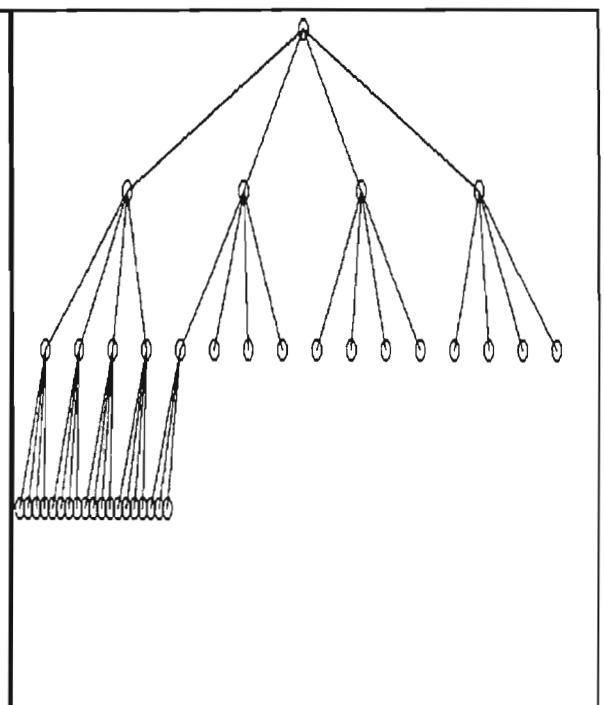


Figure 88 - Tree with Branching Factor of 4

Two star-based topologies were considered, one being the simple star which is essentially a tree with a branching factor of 39 shown in Figure 89, and one consisting of three linked stars, with the three centre stations being fully-connected, as shown in Figure 90.

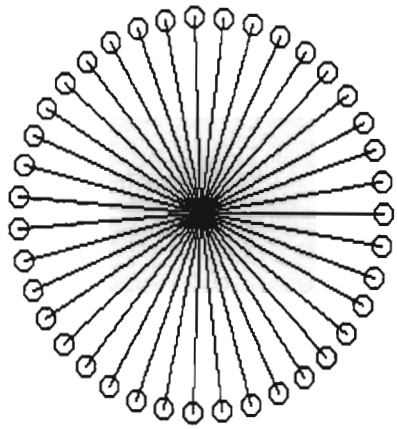


Figure 89 - A Simple Star

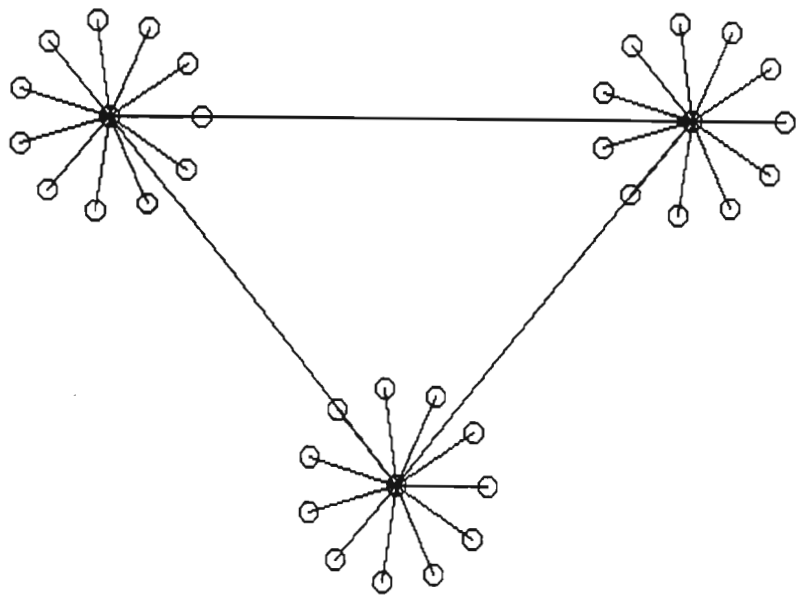


Figure 90 - Three Linked Stars

To ascertain the effect of 'bottlenecks', a network with two fully-connected twenty station components, and four links connecting four distinct vertices in the one component with four distinct vertices in the other component, was generated and then tested.

In addition to these topologies, twenty networks with connectivities from two to thirty-nine (in steps of two until fully-connected) were generated in order to determine how delay in a network would alter as connectivity increased.

These networks are the 'Harary graphs' of even connectivity over n vertices, with n here being 40, and are constructed as follows :

Let m be the connectivity required for the network, and take r to be $m/2$. Let the vertices be labelled $0,1,...,n-1$. Then, for each pair of distinct vertices j and k, a jk-edge is placed iff $j - r \leq k \leq j + r$, with addition modulo n.

This construction guarantees that each of the generated networks will not only be m-connected, but will be m-connected with the least possible number of edges necessary to achieve this [66].

6.3 Results

The routing strategy employed is not a factor in delay for trees, as there is only one available route between any two stations. Figure 91 shows the delay and deviation of delay experienced in the three trees considered, and the simple star (tree with branching factor of thirty-nine).

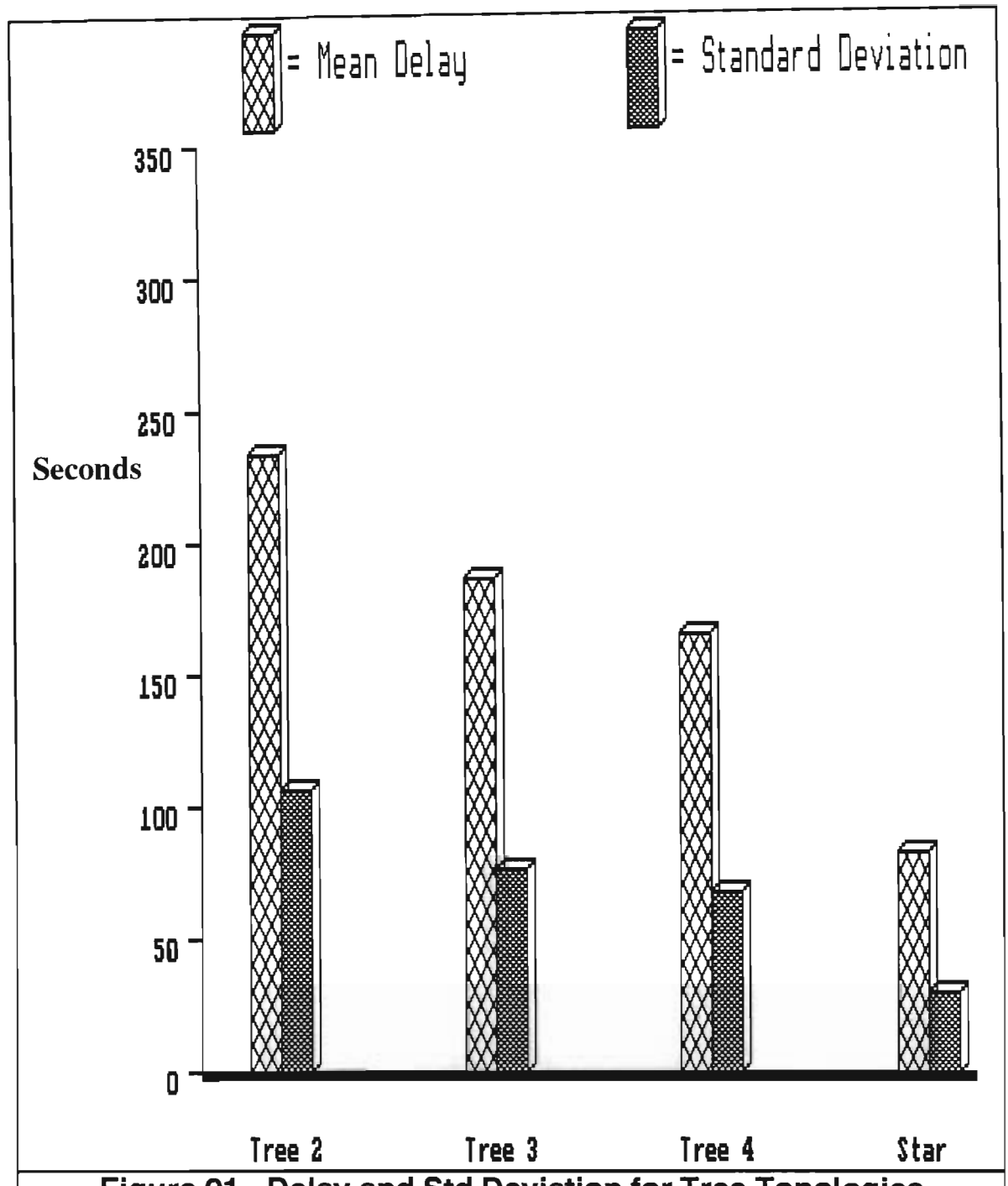


Figure 91 - Delay and Std Deviation for Tree Topologies

Clearly the higher the branching factor of the tree the less delay in the network. This result is intuitively obvious when one considers that lower branching factors will ensure greater depth of trees; and that, for the average case, the number of intermediate stations a message needs to pass through will be directly proportional to the depth of the tree.

Results for the linked star shown in Figure 90 are shown below :

Routing Strategy	Mean Delay	Std Deviation
Minimum-hop	104.05	43.18
Flooding	103.87	40.74

Clearly the routing strategy employed did not have any significant impact on delay time for this topology. This is to be expected as the flood route must be identical to the minimum-hop route in all respects, bar the subpath between the three centre stations, where flooding may choose the two-link path between two stations rather than the direct link. Clearly, however, in most cases less delay will be experienced over the direct link, and only rarely will flooding find a shorter delay path over two links with a mean delay double that of the direct link path.

The extra intermediaries in this linked-star topology makes it less effective than the simple star (104 seconds as against 82 seconds mean delay), but still significantly better than the three trees considered.

The network with two fully-connected twenty-station components linked together by four links shows a very different picture with respect to delays experienced using different routing strategies:

Delays in the 'Bottleneck' network :

Routing Strategy	Mean Delay	Std Deviation
Minimum-hop	76.66	45.29
Flooding	6.31	6.93

Firstly, considering just the minimum-hop results, we find that delay is in the same order as that experienced in the simple star. This is to be expected - apart from the special case of a message emanating from or going to the centre station, a message in a simple star will have one intermediary station, the centre station, to pass through

in going from source to destination. In our two-component network, about half the time messages will be passed direct from source to destination with no intermediary stations (source and destination in same component), and about half the time there will be two intermediary stations (source and destination in different components, in which case the message must travel from source station to one of the four 'gateway' stations in its component to one of the four 'gateways' in the other component, and finally to the destination). Again there will be occasional special cases when either the source station, the destination station, or both, are 'gateway' stations. Essentially though we will have zero intermediaries for about half the messages and two intermediaries for the rest, averaging out at one intermediary, the same as the simple star.

The gain to be had in using flood routing rather than minimum-hop is pronounced. This can be ascribed to two factors. Firstly, while minimum-hop will be using paths with the least mean delay, the fact that there is deviation associated with delay means that these paths will not always be the paths of least delay **for a particular send**. With the high degree of connectivity within components flood routing has a plethora of potential routes available, and it would be unlikely for it not to find at least one of these which was superior to the minimum-hop path.

To clarify by example, say we have a message where source and destination were in the same component. Minimum-hop would choose the direct link, and could have a delay of anywhere between 0 seconds and $20 + 14\sqrt{3}$ seconds. Say it gets an average delay of twenty seconds for a particular send. On the same send, the worst flooding can do is to arrive on the direct link after the same twenty seconds. However, in addition it has 18 two-link paths to consider, 18 x 17 three-links paths, 18 x 17 x 16 four-link paths, and so on. If on any one of these paths the total delay on links is under twenty seconds, the flood will be able to achieve a faster time for the send. (In

general it can be shown that a fully-connected network will have the number of alternate paths between source destination pairs being :

$$\sum_{i=1}^{n-2} \frac{(n-2)!}{(i-1)!}$$

where n is the number of stations, [59].)

Secondly, flooding allows for a 'bypass' potential in the case of an error during a send. In minimum-hop, a failed attempt to send on the direct link will generally result in considerable delay, while in flooding advantage will be taken of other routes where no such failures occur. It is clear that flooding will outperform minimum-hop/shortest-path strategies in networks where there are alternate routes to choose from, and will become increasingly more efficient relative to these other strategies as connectivity increases, and there are more routes to choose from. This result is due to the deviation of delay waiting for links to be enabled in the MBC environment, if delay between links were constant or links were continuous then of course flooding and shortest-path would obtain identical results.

Results of the simulation using all 40-node networks of even connectivity from two-connected to 38-connected, together with the fully-connected network on 40 vertices (39-connected) are shown in Figure 92, (full numeric results appear in Appendix G). What is apparent here is that, in the case of flooding, delay decreases exponentially as connectivity increases linearly. In the case of minimum-hop, delay is related to connectivity in line with the average distance between vertices in Harary graphs, ie $\frac{1}{2} + \frac{21}{k}$, [68]. In both cases a network designer can expect a significant decrease in network delay when increasing connectivity in a weakly-connected network, but is faced with a diminishing returns scenario as connectivity is increased further.

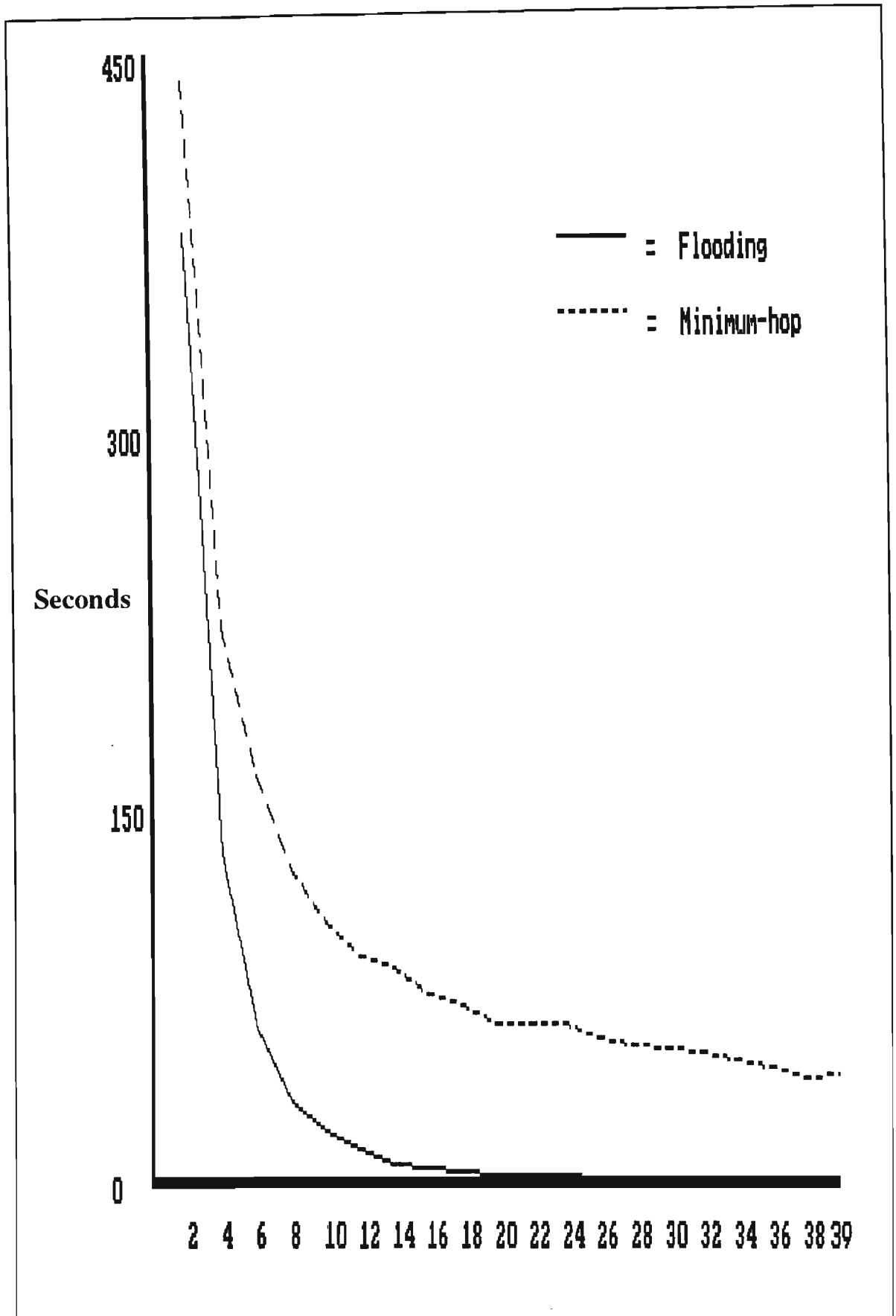


Figure 92 - Delay for Networks of Various Connectivities

The best-fit formula found for the flood results is approximately $473.4e^{-0.32.k}$, while for the minimum-hop results the formula is approximately $20 + \frac{21(40)}{k}$, where k is the connectivity of the graph. This clearly indicates that flooding's relative gain over minimum hop increases with connectivity. This is borne out by Figure 93 below, which shows flood delay as a percentage of minimum-hop delay over the various connectivities. (Results of lower than one per cent appear as zero.)

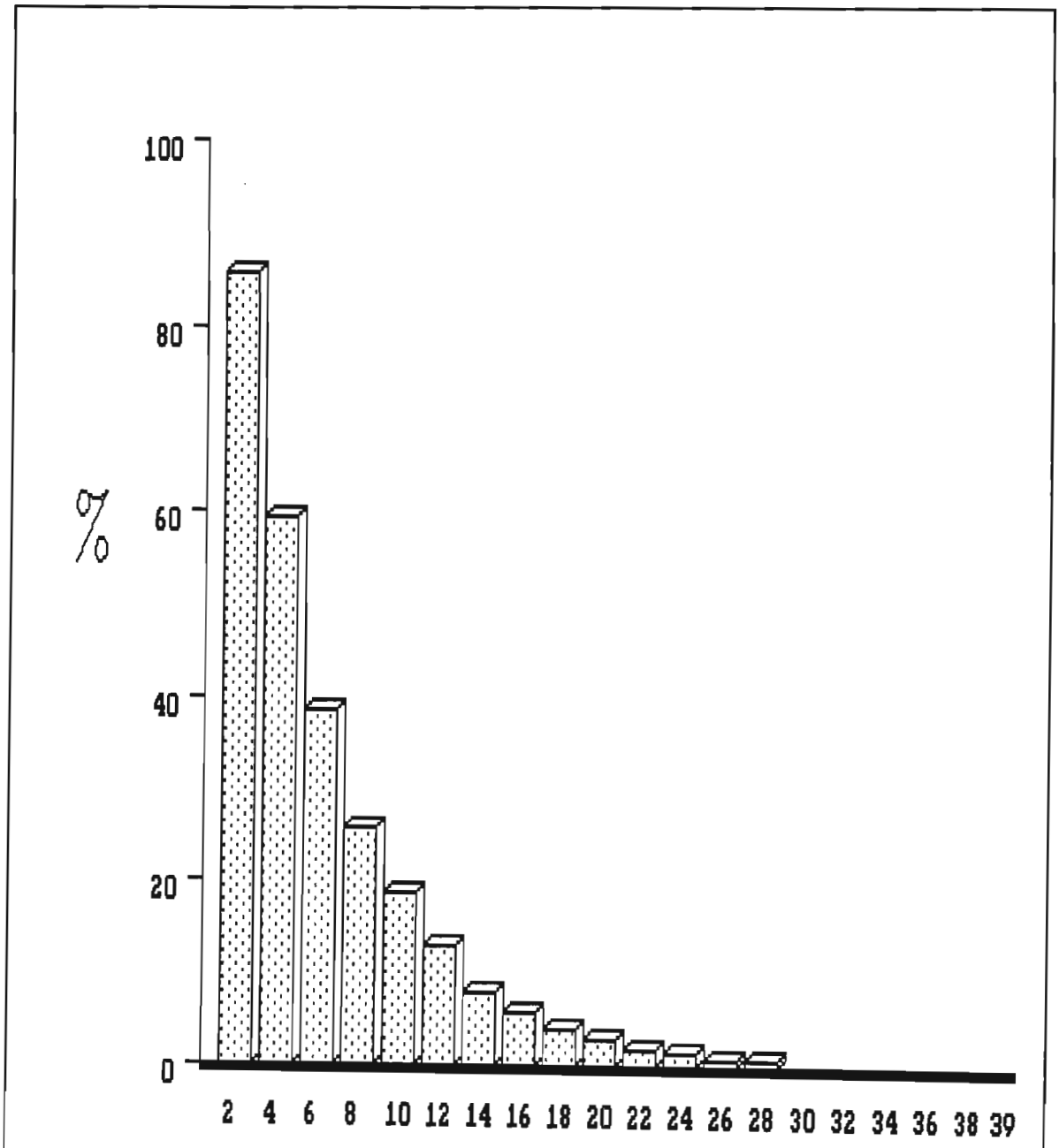


Figure 93 - Flood Delay as Percentage of Min-Hop Delay

6.4 Conclusion

The results show that, for tree-structured networks, delays will decrease as branching factors increase. Linked stars induce an extra level of intermediaries over simple stars, and so are less efficient in terms of delay.

It is clear that flooding will outperform minimum-hop/shortest- path strategies in networks where there are alternate routes to choose from, and will become increasingly more efficient relative to these other strategies as connectivity increases, and there are more routes to choose from. This result is due to the deviation of delay waiting for links to be enabled in the MBC environment, if delay between links were constant or links were continuous then of course flooding and shortest-path would obtain identical results.

Both flooding and minimum-hop/shortest-path strategies will experience initial high gains followed by significantly decreasing gains as the connectivity of networks increase. This has important implications, as it indicates that the addition of links will not only provide enhanced network integrity (ie. less vulnerability) but also more efficient performance. Clearly the gain in performance does lessen though as more and more links are added, so moving from, say, two to four connected will give a greater improvement than moving from 16 to 18 connected. Some sort of cost trade-off is clearly necessary, then, in determining how connected the network should be.

7 Conclusion

7.1 Classification of Trail Reflections

A fine classification schema has been developed and implemented. The schema is more comprehensive than others proposed, [23, 41], and results in a smaller percentage of trails being classified as belonging to the 'unknown' type.

Twenty-eight distinct types of trail reflections have been identified. It is perhaps worth mentioning here that the theoretical 'specular overdense' shape, on which much current theory is based, (see [14, 23, 49]), did not occur frequently enough to merit separate classification. In viewing around fifty thousand trails, the author has in fact encountered this shape on just three occasions.

The classification system, due to its design as a rule-based expert system, is both flexible and highly user-orientated. This allows domain experts to alter or expand the classification schema with great ease.

Initial statistics, indicating the contributions of the various families to total trail counts and durations, have been presented. As would be expected, the trail families which were recognisably underdense had the largest contribution to counts, and those that were recognisably overdense had the largest contribution to duration. However, the 'classic underdense with plateau' (type 10) and the 'hump-backed classic under-dense' (type 29) are shown to be more significant contributors to the overall under-dense counts and durations than is the 'classic underdense' (type 9) family on which most theoretical work has been based.

The 'rectified sine' (type 20) is the most important contributor to total duration.

7.2 Throughput Capacity

The overdense group, and in particular the 'rectified sine' (type 20) and 'non-sine' (type 21) families, have been identified as the major contributors to channel throughput capacity. This is true even at low data rates, such as 5kbps, and the higher the data rate the greater the contribution of these families.

There was a high degree of correlation as regards optimal data rates and block sizes between results on all trails and results on just the overdense group. This applied across all systems and simulations involved in the work, and suggests that the overdense group should be used as the basis for any theoretical model of throughput capacity in MBC, with the underdense group being used as only a slight modifying factor, if at all. In particular, results of the 'rectified sine' type have been shown to provide a good approximation to overall results, and could serve as the basis of a theoretical model.

A comparison between theoretical results as presented in [20, 49] and the measured results presented in this thesis has revealed significant differences, both in the fixed data rate case and in the adaptive data rate case. These are particularly noticeable at the lower data rates considered, which has important implications as these rates are those typically used in current MBC systems.

There did not appear to be any one optimal fixed rate across different systems, or indeed across different days on the same system. However, it is of interest that all optimal rates found were in excess of 100 kbps.

It has been shown that the use of adaptive bit rates will result in significant improvements. The 'limited-adaptive' results indicate that there will be a sharp rise in throughput as one increases maximum modem operating rate at the lower data rates,

but that returns will diminish as increase in maximum modem operating rate continues. This has major implications for the design of adaptive bit-rate modems.

7.3 Wait Time

The wait time results show clearly that far lower data rates must be employed to minimise wait time than those which must be employed to maximise throughput. All wait time data rate optimisation resulted in rates of under 25 kbps, whereas throughput optimisation yielded results of over 100 kbps.

The results show that the underdense group has a far more significant role in the wait time situation than it has in the throughput situation, although the overdense group becomes more important as message size increases.

It has been clearly demonstrated that different optimal rates are applicable for different message sizes, with higher rates for larger message sizes, as would be expected.

7.4 Networked MBC

Problems of network analysis associated with the sporadic nature of the channel have been identified.

Three basic routing strategies have been simulated in the MBC environment, with a novel algorithm for simulation of flood routing being presented.

The results clearly indicate that flood routing will ensure markedly lower delay than shortest-path or minimum-hop routing in the environment, given that the network loading is light enough to ensure minimal queuing contention. Such contention would

have to be particularly severe to make other options more attractive than flooding in highly connected networks, as here flood delay has been shown to be less than one per cent of the delays encountered using other strategies.

The effect of network topology on delay has been studied and analysed.

An important finding is that flood delay will decrease exponentially as network connectivity increases linearly, while minimum-hop delay will be alter with connectivity according to the average distance formula $\frac{1}{2} + \frac{21}{k}$. This has tremendous implications for network design, as it indicates that, regardless of the routing strategy employed, increasing connectivity (adding links) in a loosely-connected network will give marked performance improvement, but that these improvements will become less significant as the network becomes more connected.

7.5 Future Work

There is of course virtually unlimited scope for further research in this field, here only questions arising from this thesis are identified.

As regards trail classification, the same trail reflection types were encountered across all systems and links. It would be of great interest to ascertain whether these southern hemisphere results were duplicated in the northern hemisphere, or if there would be differences in the families or their contributions. This question is already being studied by individuals at Meteor Communications Corporation in Seattle, using the TrailStar classification system.

Current trail classification occurs on the basis of the entire trail, after the event. It would be tremendously useful if some schema was devised whereby classification

could be made on the basis of the early part of the trail, in real time. This, if it were feasible, would allow data rates to be selected in order to make optimal use of each individual trail.

As regards throughput capacity, the discrepancies between current theory and actual observed results presented here clearly indicate a need for further theoretical investigation in order to derive a model more compatible with the actual meteor channel characteristics.

The optimal data rates for minimising wait time using different message sizes was found to be near-identical on the two different monitoring links considered in the study. Further practical work aimed at discovering whether this applies over a wider range of systems, as well as further theoretical work geared towards discovering the cause of this similarity, is clearly indicated.

A comparison of a wider range of routing strategies than just the three discussed here would be of interest. In addition, the study of a wider range of topologies would be worthwhile.

An important piece of research would be a study of the effect of queuing contention on network delay. While this thesis has shown that queuing contention will not arise in lightly loaded networks, a formal analysis determining exactly when it would arise, and the extent of its effect as loading increases, would be of great value. Such research promises to be extremely difficult, if in fact possible, however, as the intermeshed effect of critical features such as topology and traffic flow, which are specific to individual networks, would seem to mitigate against the existence of a general solution.

Results presented here are valid for MBC networks using half-duplex mode on a single frequency. These results are not directly applicable to the full-duplex situation, in large part due to the bipartite nature of full-duplex MBC networks. (One set of stations receive on frequency A, and transmit on frequency B, the other set receives on B, and transmits on A.) Further work on the full-duplex situation is thus required.

The networks analysed here have all had fixed stations, while a number of current applications employ mobile stations. Research on the best ways to cater for mobile stations in MBC networks is thus clearly needed.

The network study here has been based on determining the effects of routing strategy, topology, and connectivity, on network performance. In order to do this, a simple ARQ-feedback scheme for error handling was assumed. However it might well be that another protocol is more effective in this environment, and research into the efficiency of different forward and feedback protocols for MBC is necessary.

Finally, a major current limitation of MBC systems is that stations must be around 500km apart for communications to be feasible. Current research work is being done into utilising ground wave transmission for closer communications. The integration of instantaneous local area ground wave networks with sporadic wide area meteor-burst networks promises to hold a number of interesting research problems.

References

[1] Nagaoka, H., *'Possibility of Disturbance of Radio Transmission by Meteor Showers'* in Proceedings of the Imperial Academy of Tokyo, Vol 5, p 632, **(1929)**.

[2] Pickard, G.W., *'A Note on the Relation of Meteor Showers and Radio Reception'*, in Proceedings of the IRE, Vol 19, pp 1166-1170, **(1931)**.

[3] Skellet, A.M., *'The Ionizing Effect of Meteors in Relation to Radio Propagation'*, in Proceedings of the IRE, Vol 20, pp 1933-1940, **(1932)**.

[4] Lovell, A.C.B., and Clegg, J.A., *'Characteristics of the Radio Echoes from Meteor Trails : I. The Intensity of the Radio Reflections and Electron Density Trails'* in Proceedings of the Physical Society of London, Vol 60, pp 491-498, **(1948)**.

[5] Lovell, A.C.B., *'Meteoric Ionisation and Ionospheric Abnormalities'* in Physical Society Progress Reports, Vol 11, pp 415-443, **(1947)**.

[6] McKinley, D.W.R., and Millman, P.M., *'A Phenomenological Theory of Radar Echoes from Meteors'* in Proceedings of the IRE, Vol 37, pp 364-375, **(1949)**.

[7] McKinley, D.W.R., *'Deceleration and Ionizing Efficiency of Meteors'* in Journal of Applied Physics, Vol 22, pp 202-213, **(1951)**.

[8] Hawkins, G.S., *'A Radio Echo Survey of Sporadic Meteor Radiants'* in Monthly Notices of the Royal Astronomical Society, Vol 116, pp 92-104, **(1956)**.

[9] Hawkins, G.S., and Brown, J.C., *'A Comprehensive Study of Meteor Echoes - I'*, Smithsonian Astrophysical Observatory Special Report 254, **(1967)**.

[10] Davis, G.W.L., Gladys, S.J., Lang, G.R., Luke, L.M., & Taylor, M.K., *'The Canadian JANET System'* in Proceedings of the IRE, Vol 45, pp 1666-1678, **(1957)**.

[11] Forsyth, P.A., Vogan, E.L., Hansen, D.R., & Hines, C.O., *'The Principles of Janet - A Meteor-Burst Communication System'* in Proceedings of the IRE, Vol 45, pp 1642-1657, **(1957)**.

[12] Hornback, C.E.; Dreyfogle, L.D., & Sugar, G.E., *'The NBS Meteor-Burst Propagation Project - A Progress Report'*, National Bureau of Standards Technical Note 86, **(1960)**.

[13] Kokjer, K.J., & Roberts, T.D., *'Networked Meteor-Burst Data Communications'* in IEEE Communications Magazine, Vol 24, pp 23-29, **(1986)**.

[14] Hellweg, G.A., *'Meteor-Burst Communications : Is This What the Navy Needs?'*, Master's Thesis, Naval Postgraduate School, Monterey, U.S.A., **(1987)**.

[15] Gilbert, G.N., *'Growth and Decline of a Scientific Specialty : The Case of Radar Meteor Research'* in Transactions of the American Geophysics Union, Vol 58, pp 273-277, **(1977)**.

[16] Sugar, G.R., *'Loss in Channel Capacity Resulting from Starting Delay in Meteor-Burst Communication'* in Journal of Research, National Bureau of Standards, Vol 64, pp 493-494, **(1960)**.

[17] Sugar, G.R., and Sullivan, K.W., *'VHF Radio Propagation Data for the Cedar Rapids-Sterling, Anchorage, Barrow, and Fargo-Churchill Test Paths, April 1951 - 1958'* in Journal of Research, National Bureau of Standards, Vol 66, pp 113-117, **(1962)**.

[18] Sugar, G.R., *'Radio Propagation by Reflection from Meteor Trails'* in Proceedings of the IEEE, Vol 52, pp 116-136, **(1964)**.

[19] Abel, M.W., *'Meteor-Burst Communications : Bits per Burst Performance Bounds'* in IEEE Transactions on Communications, Vol COM-34, pp 927-936, **(1986)**.

[20] Weitzen, J.A., Birkemeier, W.P., and Grossi, M.D., *'An Estimate of the Capacity of the Meteor-Burst Channel'* in IEEE Transactions on Communications, Vol COM-32, pp 972-974, **(1984)**.

[21] Weitzen, J.A., Grossi, M.D., and Birkemeier, W.P., *'High Resolution Multipath Measurements of the Meteor Scatter Channel'* in Radio Science, Vol 19, pp 375-381, **(1984)**.

[22] Weitzen, J.A., *'A Data Base Approach to Analysis of Meteor Burst Data'* in Radio Science, Vol 22, pp 133-140, **(1987)**.

[23] Ostergaard, J.C., Rasmussen, J.E., Sowa, M.J., Quinn, J.M., and Kossey, P.A., *'Characteristics of High-Latitude Meteor Scatter Propagation Parameters over the 45-104 MHz Band'*, paper presented at the 36th Symposium of the Electromagnetic Wave Propagation Panel (AGARD-CP-382), **(1985)**.

[24] Millstein, L.B., Schilling, D.L., Pickholtz, R.L., Sellman, J., Davidovici, S., Pavelchek, A., Schneider, A., and Eichmann, G., *'Performance of Meteor-Burst Communication Channels'* in IEEE Journal on Selected Areas of Communications, Vol SAC-5, pp 146-153, **(1987)**.

[25] Cannon, P.S., and Reed, A.P.C., *'The Evolution of Meteor-Burst Communication Systems'*, in Journal of the Institution of Electronic and Radio Engineers, Vol 57, pp 101-112, **(1987)**.

[26] Oetting, J.D., *'An Analysis of Meteor-Burst Communications for Military Applications'* in IEEE Transactions on Communications, Vol COM-38, pp 1591-1601, **(1980)**.

[27] Whittaker, C.J., *'Meteor Burst Communication'* in Journal of Research of the Signals Institute, Vol 18, pp 185-190, **(1988)**.

[28] Gray, K.G., *'Meteor Burst Communications'* in Signal, Vol 36, pp 125-134, **(1982)**.

[29] Richmond, R.L., *'Meteor Burst Communications Part 1 : MBC Advances Assist C^3 Objectives'* in Military Electronics / Countermeasures, Vol 8, pp 68-72, **(1982)**.

[30] Boyle, D., *'Long Distance Communications - Back to Ionization'* in International Defence Review, Vol 21, pp 491-493, **(1988)**.

[31] Johnon, J.H., *'Solid-state Amplifier Delivers Reliable Meteor-Burst Communication'* in Microwave System News and Communications Technologies, Vol 17, pp 8-16, **(1987)**.

[32] Gottlieb, I., *'Meteoric Bursts Could Keep Post-Attack Communications Open'* in Defense Electronics, Vol 13, pp 61-69, **(1981)**.

[33] Larsen, J.D., Meteor Communications Corporation, *personal communication*, **(1990)**.

[34] Donich, T.G., Meteor Communications Corporation, *personal communication,* **(1990)**.

[35] Barton, M., and Burke, M., *'SNOTEL - An Operational Data Acquisition System using Meteor-Burst Technology'*, paper presented at the Western Show Conference, **(1977)**.

[36] Crook, A.G., *'SNOTEL : Monitoring Climatic Factors to Predice Water Supplies'* in Journal of Soil and Water Conservation, Vol 32, pp 294-295, **(1977)**.

[37] Day, W.E., *'Meteor-Burst Communications Bounce Signals Between Remote Sites'* in Electron, Vol 55, pp 71-75, **(1982)**.

[38] Morgan, E.J., *'Meteor Burst Communications : An Update'* in Signal, Vol 42, pp 55-61, **(1988)**.

[39] Handley, P., Salbu (Pty) Ltd, *personal communication*, **(1991)**.

[40] Hernan, J.R., Duggan , C.C., Thompson, W.I., Costa, R.A., and Deluca, D.M., *'Narrowband Digital Voice Communications Over a Meteor Burst Channel'*, in Electronics Letters, Vol 23, pp 59-60, **(1987)**.

[41]Weitzen, J.A. and Tolman, S., *'Technique for Automatic Classification of Meteor Trails and Other Propagation Mechanisms for the Air Force High Latitude Meteor Burst Test Bed'*, Signatron, Inc. Report # A447-13 for Rome Air Development Center (RADC-TR-86-117), **(1986)**.

[42] Ince, A.N., *'Communications through EM-Wave Scattering'*, in IEEE Communications Magazine, Vol 20, pp 27-43, **(1982)**.

[43] Mawrey, R.S., *'A Meteor Scatter Measurement System'*, M.Sc thesis, University of Natal, Durban, **(1988)**.

[44] Hayes-Roth, F. et al. (editors), *'Building Expert Systems'*, Addison-Wesley, **(1983)**.

[45] Buchanan, B., *'New Research on Expert Systems'*, in *'Machine Intelligence 10 - Intelligent Systems'*, edited by J. Hayes et al., Halsted Press, **(1982)**.

[46] Feigenbaum, A.E., *'The Art of AI - Themes and Case Studies in Knowledge Engineering'*, in *'Auerbach Annual Best Computer Papers'*, edited by I. Auerbach, North-Holland, **(1979)**.

[47] McKinley, D.W.R., *'Meteor Science and Engineering'*, McGraw-Hill, **(1961)**.

[48] Weitzen, J.A., Meteor Communications Corporation, *personal communication*, **(1990)**.

[49] Weitzen, J.A., *'Feasibility of High Speed Digital Communications on the Meteor Scatter Channel'*, Ph.D Thesis, University of Wisconsin, Madison, **(1983)**.

[50] Davidovici, S. and Kanterakis, E.G., *'Performance of a Meteor-Burst Communication System Using Packet Messages with Variable Data Rates'*, in IEEE Transactions on Communications, Volume COM-37, pp 6 - 17, **(1989)**.

[51] Pursley, M.B. and Sandberg, S.D., *'Variable-Rate Coding For Meteor-Burst Communications'*, in IEEE Transactions on Communications, Volume COM-37, pp 1105 - 1112, **(1989)**.

[52] Melville, S.W., Larsen, J.D., Letschert, R.Y., and Goddard, W.D., *'The Classification of Meteor Trail Reflections by a Rule-Based System'*, in Transactions of the SAIEE, Volume 80, pp 104 - 116, **(1989)**.

[53] Schwartz, M., *'Information Transmission, Modulation and Noise"*, Tokyo, McGraw-Hill, **(1981)**.

[54] Winston, P., *'Artificial Intelligence'*, Reading, Mass., Addison-Wesley, **(1981)**.

[55] Larsen, J.D., Melville, S.W., Mawrey, R.S., Letschert, R.Y., and Goddard, W.D. *'Throughput Capacity of Meteor Burst Communications'* in Transactions of the SAIEE, Vol 81, pp 20 - 30, **(1990)**.

[56] Melville, S.W., *'Observations on the Relative Importance of Different Trail Families in Meteor Burst Communications'* in Proceedings of the Third South African IEEE Conference on Communications and Signal Processing, pp 26 - 32, **(1990)**.

[57] Larsen, J.D., Melville, S.W., and Mawrey,R.S.M., *'Adaptive Data Rate Capacity of Meteor-Burst Communications'* in Conference Record of the 1990 IEEE Conference on Military Communications, Vol 2, pp 40.1.1 - 40.1.5, **(1990)**.

[58] Melville, S.W., *'Analysing Routing Strategies in Sporadic Networks'* in Proceedings of the 6th Southern African Computer Symposium, pp 148-166, (since also invited for publication in the South African Computer Journal), **(1991)**.

[59] Melville, S.W., and Carson, D.I., *'Networking Meteor-Burst Communications'* in Proceedings of the Fourth South African IEEE Conference on Communications and Signal Processing, pp 132-137, **(1991)**.

[60] Melville, S.W. and Larsen, J.D., *'Wait Time in Meteor-Burst Communications'* to be published in Transactions of the SAIEE, **(1992)**.

[61] Corson, S. and Ephermides, A., *'A Distributed Routing Algorithm for Mobile Radio Networks'* in Conference Record of the 1989 IEEE Conference on Military Communications, Vol 1, pp 11.2.1 - 11.2.4., **(1989)**.

[62] Larsen, J.D., Meteor Communications Corporation, *personal communication*, **(1990)**.

[63] Djikstra, E.W., *'A Note on Two Problems in Connection with Graphs'* , in Numerische Mathematik I, pp 269-271, **(1959)**.

[64] Lawler, E.W. and Wood, D.E., *'Branch-and-Bound Methods : A Survey'* , in Operations Research, Vol 14, pp 669-719, **(1966)**.

[65] Ibaraki, T., *'The Power of Dominance Relations in Branch-and-Bound Search Methods'*, in JACM, Vol 24, pp 264-279, **(1977)**.

[66] Harary, F., *'The Maximum Connectivity of a Graph'* in Proceedings of the National Academy of Science, U.S.A., Vol 48, pp 1142-1146, **(1962)**.

[67] Taub, H. and Schilling,D.L., *'Principles of Communications'* (second edition), Tokyo, McGraw-Hill, p460, **(1986)**.

[68] Goddard, W.D., Massauchussetts Institute of Technology, *personal communication*, **(1991)**.

Appendix A - Listing of TrailStar Condition Base

Note : Items appearing after '!' symbols are replaced by parameters.

```
                                                                        *
-------------------------------------------------------------------------
Condition 1
 Dummy condition to trigger default rule if all else fails.
                                                                        *
-------------------------------------------------------------------------
Condition 2
 Straight line variance over trail > !<number> times
variance over fall.
                                                                        *
-------------------------------------------------------------------------
Condition 3
 Straight line variance over trail <= !<number> times
variance over fall.
                                                                        *
-------------------------------------------------------------------------
Condition 4
 Straight line variance over trail > !<number> times
variance over rise.
                                                                        *
-------------------------------------------------------------------------
Condition 5
 Straight line variance over trail <= !<number> times
variance over rise.
                                                                        *
-------------------------------------------------------------------------
Condition 6
 Trail duration greater than !<number> times 50 mSec.
                                                                        *
-------------------------------------------------------------------------
Condition 7
 Trail duration less than or equal to !<number> times 50 mSec.
                                                                        *
-------------------------------------------------------------------------
Condition 8
 Upper plateau is present.
                                                                        *
-------------------------------------------------------------------------
Condition 9
 Upper plateau is not present.
                                                                        *
-------------------------------------------------------------------------
Condition 10
 Peak is in !<Xth> third of trail.
                                                                        *
-------------------------------------------------------------------------
Condition 11
 Amplitude range over trail is greater
 than !<number> dBm.
                                                                        *
-------------------------------------------------------------------------
Condition 12
 Amplitude range over trail is less
 than or equal to !<number> dBm.
                                                                        *
-------------------------------------------------------------------------
Condition 13
 Peak strength is greater than ( !<number> - 130) dBm.
                                                                        *
-------------------------------------------------------------------------
Condition 14
 Peak strength is less than or equal to ( !<number> - 130) dBm.
                                                                        *
-------------------------------------------------------------------------
Condition 15
 Time constant on fall greater than !<number> times 0.01.
                                                                        *
-------------------------------------------------------------------------
```

```
*
Condition 16
 Time constant on fall is less than or equal
 to !<number> times 0.01.
*
Condition 17
 Straight line variance is greater than !<number> times 0.1.
*
Condition 18
 Straight line variance is less than or equal
 to !<number> times 0.1.
*
Condition 19
 Upper plateau comprises greater than !<number> % of trail.
*
Condition 20
 Upper plateau comprises less or equal to !<number> % of trail.
*
Condition 21
 There are more than !<number> upper plateaus.
*
Condition 22
 There are less than or equal to !<number> upper plateaus.
*
Condition 23
 There are > !<number> fades, each of which
 is > !<number> * 10 mSec.
*
Condition 24
 There are <= !<number> fades, each of which
 is > !<number> * 10 mSec.
*
Condition 25
 Trail has one or more values outside reasonable limits.
*
Condition 26
 Peak is in !<Xth> !<number> % of trail.
*
Condition 27
 Peak is not in !<Xth> !<number> % of trail.
*
Condition 28
 Fall good in last !<number> % of fall.
*
Condition 29
 Fall not good in last !<number> % of fall.
*
Condition 30
 Rise slope - ABS(fall slope) <= !<number> * 0.01.
 ("closeness degree").
*
Condition 31
 Rise slope - ABS(fall slope) > !<number> * 0.01.
*
```

```
*
Condition 32
 ABS( line slope over trail) <= !<number> * 0.01.
 ("horizontalness").
*
Condition 33
 ABS( line slope over trail) > !<number> * 0.01.
*
Condition 34
 Intersection of rise and fall slopes > !<number> dBm
 above peak.
*
Condition 35
 Intersection of rise and fall slopes <= !<number> dBm
 above peak.
*
Condition 36
 Rise segments with tolerance !<number> * 0.01
 are <= !<number>.
*
Condition 37
 Rise segments with tolerance !<number> * 0.01 are <= !<number>.
*
Condition 38
 Trail segments with tolerance !<number> * 0.1 is > !<number>.
*
Condition 39
 Trail segments with tolerance !<number> is <= !<number>.
*
Condition 40
 Segments calculated above with
 slope < ABS( !<number> * 0.01) is > !<number>.
*
Condition 41
 Segments calculated above with
 slope < ABS( !<number> * 0.01) <= !<number>.
*
Condition 42
 Lines calculated above consistent with twin-type
 rise,fall,rise,fall.
*
Condition 43
 Absolute minimum in !<Xth> !<number> % of trail.
*
Condition 44
 Absolute minimum not in !<Xth> !<number> % of trail.
*
Condition 45
 Absolute minimum before peak.
*
Condition 46
 Absolute minimum after peak.
*
Condition 47
 Absolute minimum significantly more pronounced
 than other minima.
*
```

```
*
Condition 48
 Absolute minimum not significantly more pronounced
 than other minima.
*
Condition 49
 Peak near centre of upper plateau
 comprising > !<number> % of trail.
*
Condition 50
 Peak not near centre of upper plateau
 comprising > !<number> % of trail.
*
Condition 51
 Variance from parabola is greater than !<number> * 0.1.
*
Condition 52
 Variance from parabola is less than or
 equal to !<number> * 0.1.
*
Condition 53
 Above !<number> % parabola between minima
 with variance <= !<number> * 0.1
*
Condition 54
 Below !<number> % parabola between minima
 with variance <= !<number> * 0.1
*
Condition 55
 There are greater than !<number> local extrema.
*
Condition 56
 There are less than !<number> local extrema.
*
Condition 57
 Time since last trail greater than !<number> * 0.1 seconds.
*
Condition 58
 Time since last trail less than or equal
 to !<number> * 0.1 seconds.
*
Condition 59
 Last trail sample mean within !<number> dBm of current mean.
*
Condition 60
 Last trail sample mean not within !<number> dBm
 of current mean.
*
Condition 61
 Duration of preceding trail greater than !<number> * 50 mSec.
*
Condition 62
 Duration of preceding trail less than or equal
 to !<number> * 50 mSec.
*
```

```
*
Condition 63
 Above !<number> % parabola between maxima with
 variance <= !<number> * 0.01
*
Condition 64
 Below !<number> % parabola between maxima with
 variance <= !<number> * 0.1
*
Condition 65
 A parabola of variance <= !<number> * 0.1 covers
 50% of the trail.
*
Condition 66
 No parabola of variance <= !<number> * 0.1 covers
 50% of trail.
*
Condition 67
 Variance from straight line on fall > !<number> * 0.1.
*
Condition 68
 Variance from straight line on fall <= !<number> * 0.1.
*
Condition 69
 Variance from straight line on rise > !<number> * 0.1.
*
Condition 70
 Variance from straight line on rise <= !<number> * 0.1.
*
Condition 71
 Fall time is greater than plateau time.
*
Condition 72
 Fall time is less than or equal to plateau time.
*
Condition 73
 Line segments needed over fall with
 tolerance !<number> is > !<number>.
*
Condition 74
 Line segments needed over fall with
 tolerance !<number> is <= !<number>.
*
Condition 75
 Line segments over trail with
 tolerance !<number> equals !<number>.
*
Condition 76
 Line segments calculated above rise for more
 than !<number> % of fall.
*
Condition 77
 Line segments calculated above rise for
 <= !<number> % of fall.
*
```

```
                                                                    *
Condition 78
 Rise slope duration greater than !<number> * 5 mSec.
                                                                    *
Condition 79
 Rise slope duration less than or equal to !<number> * 5 mSec.
                                                                    *
Condition 80
 Curve coefficient of parabola over trail is > !<number> - 80.
                                                                    *
Condition 81
 Curve coefficient of parabola over trail is <= !<number> - 80.
                                                                    *
Condition 82
 Good parabola fit from most pronounced
 local min. on fall to end of trail.
                                                                    *
Condition 83
 Bad parabola fit from most pronounced
 local min. on fall to end of trail.
                                                                    *
```

Explanation of 'non-obvious' conditions.

Some detail is provided below on those conditions whose workings might not be obvious.

Condition 8 : Upper plateau is present
An upper plateau is determined to be present if four or more consecutive amplitude samples are within 2 dBm of the trail's peak amplitude.

Condition 9 : Upper plateau is not present
The inverse of condition 8

Condition 25 : Trail has one or more values outside reasonable limits.
This test will return true if any sample in the trail has amplitude greater than -80 dBm, or if the duration of the trail is not in the range 15ms to 50 000 ms.

Condition 42 : Lines calculated above consistent with twin-type rise, fall, rise, fall.
This test will return true if the lines calculated by a previously checked line segment testing routine (condition 73, 74 or 75) consist of exactly four lines, with the first and third of these having a positive slope and the second and fourth a negative one.

Condition 47 : Absolute minimum significantly more pronounced than other minima.
This holds if all other minima, apart from those within ten percent of the trail duration on either side of the absolute minimum, are at least 10 dBm higher than the absolute minimum. (The reason for only checking against minima not 'close' to the absolute is to avoid this condition failing due to a subsidiary 'spike' in the fall to or rise from the absolute minimum.

Condition 48 : Absolute minimum not significantly more pronounced than other minima.
Inverse of condition 47.

Condition 55 : There are greater than !<number> local extrema.
There is a tolerance of 4 dBm in either direction to determine whether a point is an extremum or not. For example, say a local minimum had been the last extremum found. The system will scan consecutive samples, keeping track of the highest value sample encountered. Once there is greater than a 4 dBm gap between this value and the current sample encountered, the number of maxima will increase by one, and the search for extrema restarts. (This time, of course, checking for a minimum.)

Condition 63 : Above !<number> parabolas between maxima with variance < ! <number> * 0.01
More than !<number> percent of the parabola fits between maxima were fitted with a variance of less than !<number> * 0.01 (ie. were 'good' fits)

Condition 64 : Below !<number> parabolas between maxima with variance < !<number>* 0.01
Inverse of condition 63

Condition 65 : A parabola of variance < !<number> * 0.1 covers 50% of the trail.
A 'good' (variance less than user-supplied parameter) parabola fit can be made over an area comprising more than half the trail.

Condition 66 : No parabola of variance < !<number> * 0.1 covers 50% of the trail.
Inverse of condition 65.

Condition 73 : Line segments needed over fall with tolerance !<number> is <= !<number>.
Number of line segments needed to cover entire fall region is less than or equal to some user-supplied number. Line segments are constructed by doing line fits, point by point, beginning new lines when the variance between the current best line fit and the actual samples the line is being fitted to is greater than the user-specified tolerance.

Condition 74 : Line segments needed over fall with tolerance !<number> is > !<number>.
Inverse of condition 73

Condition 75 : Line segments over trail with tolerance !<number> equals !<number>.
As in condition 73, here however lines are fit to the whole trail rather than just the fall region, and the test is for equality.

Condition 76 : Line segments calculated above rise for more than !<number> % of fall.
This condition should clearly only be used after a prior call to condition 73 or 74, which will determine line fits needed in the fall region. It tests whether more than the user-defined percentage of the lines fitted have positive slopes.

Condition 77 : Line segments calculated above rise for less than !<number> % of fall.
Inverse of condition 76.

Condition 80 : Curve coefficient of parabola over trail is greater than !<number> - 80.
A parabola fit is made over the trail. A parabola is defined by Ax + Bx + C, the curve coefficient used here is defined as A times (number of samples). (Testing how 'rounded' the parabola is.)

Condition 81 : Curve coefficient of parabola over trail is less than or equal to !<number> - 80.
Inverse of condition 80.

Condition 82 : Good parabola fit from most pronounced local minimum on fall to end of trail.
Calculate curve coefficient as in condition 80, but here only using the region of the trail between the most pronounced local minimum and the end of the trail (if the most pronounced local minimum is in fact at the end of the trail the condition returns false). Now, if the fit has variance greater than 1.2 (not a good fit) then the condition returns false. Otherwise, a check is made as to the 'flatness' of the fit. (Don't want to call straight lines parabolas here.) If the curve coefficient is less than or equal to -5 then the condition returns true ('rounded' enough) else it returns false.

Condition 83 : Bad parabola fit from most pronounced local minimum on fall to end of trail.
Inverse of condition 82.

Appendix B - Listing of TrailStar Action Base

```
*
Action 1
 Trail typed as UNREASONABLE DATA.
*
Action 2
 Trail typed as SHORT, MID-PEAK.
*
Action 3
 Trail typed as SHORT MUSH.
*
Action 4
 Trail typed as MEDIUM-TIME MID-PEAK.
*
Action 5
 Trail typed as FLAT CLASSIC ("First quintile low classic")
*
Action 6
 Trail typed as FLAT BELL.
*
Action 7
 Trail typed as STRAIGHT-LINE MUSH, MEDIUM LENGTH.
*
Action 8
 Trail typed as STRAIGHT-LINE MUSH, LONG LENGTH.
*
Action 9
 Trail typed as CLASSIC UNDERDENSE.
*
Action 10
 Trail typed as CLASSIC UNDERDENSE WITH PLATEAU.
*
Action 11
 Trail typed as ROUND-TOP CLASSIC UNDERDENSE.
*
Action 12
 Trail typed as CLASSIC UNDERDENSE WITH LATE FALL.
*
Action 13
 Trail typed as CLASSIC UNDERDENSE WITH NOTCHED RISE.
*
Action 14
 Trail typed as CLASSIC UNDERDENSE WITH BAD ("Drunk") RISE.
*
Action 15
 Trail typed as BELL.
*
Action 16
 Trail typed as MULTI-PLATEAU UNDERDENSE.
*
Action 17
 Trail typed as MULTI-SLOPE UNDERDENSE.
*
Action 18
 Trail typed as TWINS.
*
```

*

Action 19
Trail typed as SQUARE ROOT SIGN.

*

Action 20
Trail typed as RECTIFIED SINE WAVE OVERDENSE.

*

Action 21
Trail typed as NON-SINE OVERDENSE.

*

Action 22
Trail typed as GOTHIC ROCKER. (Weird, unknown type.)

*

Action 23
Trail typed as DOWNWARD-TENDING STRAIGHT-LINE MUSH.

*

Action 24
Trail typed as HAZY CLASSIC.

*

Action 25
Trail typed as EXTENSION MUSH.

*

Action 26
Trail typed as EXTENSION OVERDENSE.

*

Action 27
Trail typed as SINUSOIDAL OVERDENSE.

*

Action 28
Trail typed as WIND-BLOWN OVERDENSE.

*

Action 29
Trail typed as HUMP-BACKED CLASSIC.

*

Appendix C - Listing of TrailStar Rule Base

Important Note : There is a certain amount of 'fall-through' in the rule base presented below - in other words, a rule may be determined true not merely by the fact that its conditions are true, but also because rules tested earlier have failed. The order of rule testing (priority order) of the rule base is thus important. In this base the rules are tested in the following sequence : 1, 23, 24, 2, 3, 4, 6, 7, 8, 19, 9, 28, 10, 11, 20, 12, 16, 29, 13, 5, 17, 26, 18, 27, 21, 22, 25, 14, 30, 31, 32, 15.

```
                                                                    *
CONDITIONS OF RULE #1
[1] Trail has one or more values outside reasonable limits.
ACTION OF RULE #1
 Trail typed as UNREASONABLE DATA.
                                                                    *
CONDITIONS OF RULE #2
[1] Trail duration less than or equal to 2 times 50 mSec.
[2] Straight line variance over trail > 3 times
     variance over fall.
[3] Straight line variance over trail > 3 times
      variance over rise.
[4] Peak is in 2nd third of trail.
[5] Amplitude range over trail is greater than 3 dBm.
ACTION OF RULE #2
 Trail typed as SHORT, MID-PEAK.
                                                                    *
CONDITIONS OF RULE #3
[1] Trail duration less than or equal to 1 times 50 mSec.
ACTION OF RULE #3
 Trail typed as SHORT MUSH.
                                                                    *
CONDITIONS OF RULE #4
[1] Trail duration greater than 2 times 50 mSec.
[2] Trail duration less than or equal to 20 times 50 mSec.
[3] Straight line variance over trail > 3 times
     variance over fall.
[4] Straight line variance over trail > 3 times
     variance over rise.
[5] Peak is in 2nd  third of trail.
[6] Upper plateau is not present.
ACTION OF RULE #4
 Trail typed as MEDIUM-TIME MID-PEAK.
                                                                    *
CONDITIONS OF RULE #5
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st  20 % of trail.
[3] Straight line variance over trail > 1 times
     variance over fall.
[4] Straight line variance over trail > 1 times
     variance over rise.
[5] Amplitude range over trail is less than or
     equal to 10 dBm.
[6] Straight line variance is less than or equal to
     15 times 0.1.
[7] Variance from straight line on fall <= 10 * 0.1.
[8] Variance from straight line on rise <= 10 * 0.1.
ACTION OF RULE #5
 Trail typed as FLAT CLASSIC ("First quintile low classic").
                                                                    *
```

*

CONDITIONS OF RULE #6
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Rise slope - ABS(fall slope) <= 5 * 0.01.
("closeness degree").
[3] Straight line variance is greater than 8 times 0.1.
[4] Amplitude range over trail is less than or equal
to 6 dBm.
[5] Rise slope duration greater than 0 * 5 mSec.
ACTION OF RULE #6
Trail typed as FLAT BELL.

*

CONDITIONS OF RULE #7
[1] Trail duration less than or equal to 10 times 50 mSec.
[2] Straight line variance is less than or
equal to 8 times 0.1.
[3] ABS(line slope over trail) <= 10 * 0.01.
("horizontalness").
ACTION OF RULE #7
Trail typed as STRAIGHT-LINE MUSH, MEDIUM LENGTH.

*

CONDITIONS OF RULE #8
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Straight line variance is less than or
equal to 8 times 0.1.
[3] ABS(line slope over trail) <= 10 * 0.01.
("horizontalness").
ACTION OF RULE #8
Trail typed as STRAIGHT-LINE MUSH, LONG LENGTH.

*

CONDITIONS OF RULE #9
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st third of trail.
[3] Straight line variance over trail > 3 times
variance over fall.
[4] Straight line variance over trail > 2 times
variance over rise.
[5] Amplitude range over trail is greater than 3 dBm.
[6] Straight line variance is greater than 8 times 0.1.
[7] Variance from straight line on fall <= 10 * 0.1.
[8] Variance from straight line on rise <= 10 * 0.1.
[9] Upper plateau comprises less or equal to 17 % of trail.
ACTION OF RULE #9
Trail typed as CLASSIC UNDERDENSE.

*

*

CONDITIONS OF RULE #10

[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st 66 % of trail.
[3] Straight line variance over trail > 3 times variance over fall.
[4] Straight line variance over trail > 2 times variance over rise.
[5] Amplitude range over trail is greater than 3 dBm.
[6] Straight line variance is greater than 5 times 0.1.
[7] Peak not near centre of upper plateau comprising > 15 % of trail.
[8] Variance from straight line on fall <= 10 * 0.1.
[9] Variance from straight line on rise <= 10 * 0.1.
[10] Fall time is greater than plateau time.

ACTION OF RULE #10

Trail typed as CLASSIC UNDERDENSE WITH PLATEAU.

*

CONDITIONS OF RULE #11

[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st 66 % of trail.
[3] Straight line variance over trail > 3 times variance over fall.
[4] Straight line variance over trail > 2 times variance over rise.
[5] Amplitude range over trail is greater than 3 dBm.
[6] Straight line variance is greater than 5 times 0.1.
[7] Peak near centre of upper plateau comprising > 15 % of trail.
[8] Variance from straight line on fall <= 10 * 0.1.
[9] Variance from straight line on rise <= 10 * 0.1.
[10] Fall time is greater than plateau time.
[11] Fall time is greater than plateau time.

ACTION OF RULE #11

Trail typed as ROUND-TOP CLASSIC UNDERDENSE.

*

CONDITIONS OF RULE #12

[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st third of trail.
[3] Straight line variance over trail > 2 times variance over rise.
[4] Straight line variance over trail <= 3 times variance over fall.
[5] Amplitude range over trail is greater than 3 dBm.
[6] Straight line variance is greater than 5 times 0.1.
[7] Upper plateau comprises less or equal to 15 % of trail.
[8] Variance from straight line on rise <= 10 * 0.1.
[9] Fall good in last 80 % of fall.

ACTION OF RULE #12

Trail typed as CLASSIC UNDERDENSE WITH LATE FALL.

*

```
                                                                        *
CONDITIONS OF RULE #13
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st third of trail.
[3] Straight line variance over trail > 3 times
     variance over fall.
[4] Amplitude range over trail is greater than 3 dBm.
[5] Straight line variance is greater than 5 times 0.1.
[6] Straight line variance over trail <= 2 times
     variance over rise.
[7] Upper plateau comprises less or equal to 15 % of trail.
[8] Variance from straight line on fall <= 10 * 0.1.
ACTION OF RULE #13
 Trail typed as CLASSIC UNDERDENSE WITH BAD ("Drunk") RISE.
                                                                        *
CONDITIONS OF RULE #14
[1] Trail duration greater than 20 times 50 mSec.
[2] Amplitude range over trail is greater than 15 dBm.
ACTION OF RULE #14
 Trail typed as NON-SINE OVERDENSE.
                                                                        *
CONDITIONS OF RULE #15
[1]Dummy condition to trigger default rule if all else fails.
ACTION OF RULE #15
 Trail typed as GOTHIC ROCKER. (Weird, unknown type.)
                                                                        *
CONDITIONS OF RULE #16
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st third of trail.
[3] Straight line variance over trail > 3 times
     variance over fall.
[4] Amplitude range over trail is greater than 3 dBm.
[5] Straight line variance is greater than 8 times 0.1.
[6] Straight line variance over trail <= 2 times
     variance over rise.
[7] Rise segments with tolerance 6 * 0.01 are <= 2.
[8] Variance from straight line on fall <= 10 * 0.1.
ACTION OF RULE #16
 Trail typed as CLASSIC UNDERDENSE WITH NOTCHED RISE.
                                                                        *
CONDITIONS OF RULE #17
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Straight line variance over trail > 2 times
     variance over rise.
[3] Straight line variance over trail <= 3 times
     variance over fall.
[4] There are less than 5 local extrema.
[5] Variance from straight line on rise <= 10 * 0.1.
[6] Line segments needed over fall with tolerance 6 is > 1.
[7] Line segments calculated above rise for <= 10 % of fall.
[8] Segments calculated above with
     slope < ABS( 5 * 0.01) is > 0.
ACTION OF RULE #17
 Trail typed as MULTI-PLATEAU UNDERDENSE.

                                                                        *
```

*

CONDITIONS OF RULE #18
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Straight line variance over trail > 2 times
variance over rise.
[3] Straight line variance over trail <= 3 times
variance over fall.
[4] There are less than 5 local extrema.
[5] Variance from straight line on rise <= 10 * 0.1.
[6] Line segments needed over fall with
tolerance 6 is > 1.
[7] Line segments calculated above rise for <= 10 % of fall.
[8] Segments calculated above with
slope < ABS(5 * 0.01) <= 0.
ACTION OF RULE #18
Trail typed as MULTI-SLOPE UNDERDENSE.

*

CONDITIONS OF RULE #19
[1] Trail duration less than or equal to 100 times 50 mSec.
[2] Amplitude range over trail is greater than 7 dBm.
[3] Line segments over trail with tolerance 20 equals 4.
[4] Lines calculated above consistent with twin-type
rise,fall,rise,fall.
ACTION OF RULE #19
Trail typed as TWINS.

*

CONDITIONS OF RULE #20
[1] Absolute minimum significantly more pronounced than
other minima.
[2] Absolute minimum before peak.
[3] Absolute minimum in 1st 35 % of trail.
[4] Amplitude range over trail is greater than 15 dBm.
ACTION OF RULE #20
Trail typed as SQUARE ROOT SIGN.

*

CONDITIONS OF RULE #21
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Straight line variance is greater than 8 times 0.1.
[3] Variance from parabola is less than or equal to 12 * 0.1.
[4] Peak is not in 1st 20 % of trail.
[5] Peak is not in 5th 20 % of trail.
[6] Curve coefficient of parabola over trail is <= 70 - 80.
ACTION OF RULE #21
Trail typed as BELL.

*

CONDITIONS OF RULE #22
[1] Amplitude range over trail is greater than 9 dBm.
[2] Above 25 % parabola between minima with
variance <= 12 * 0.1
ACTION OF RULE #22
Trail typed as RECTIFIED SINE WAVE OVERDENSE.

*

*

CONDITIONS OF RULE #23
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Straight line variance is less than or
equal to 8 times 0.1.
ACTION OF RULE #23
Trail typed as DOWNWARD-TENDING STRAIGHT-LINE MUSH.

*

CONDITIONS OF RULE #24
[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Straight line variance is greater than 8 times 0.1.
[3] Straight line variance over trail > 1 times
variance over fall.
[4] Straight line variance over trail > 1 times
variance over rise.
[5] Peak is in 1st 20 % of trail.
[6] Variance from straight line on fall <= 15 * 0.1.
[7] Variance from straight line on rise <= 15 * 0.1.
ACTION OF RULE #24
Trail typed as HAZY CLASSIC.

*

CONDITIONS OF RULE #25
[1] Time since last trail less than or equal
to 10 * 0.1 seconds.
[2] Last trail sample mean within 10 dBm of current mean.
[3] Trail duration less than or equal to 10 times 50 mSec.
ACTION OF RULE #25
Trail typed as EXTENSION MUSH.

*

CONDITIONS OF RULE #26
[1] Time since last trail less than or equal
to 10 * 0.1 seconds.
[2] Last trail sample mean within 10 dBm of current mean.
[3] Trail duration greater than 10 times 50 mSec.
[4] Duration of preceding trail greater than 10 * 50 mSec.
ACTION OF RULE #26
Trail typed as EXTENSION OVERDENSE.

*

CONDITIONS OF RULE #27
[1] Amplitude range over trail is greater than 9 dBm.
[2] Above 25 % parabola between minima with
variance <= 12 * 0.1
[3] Above 25 % parabola between maxima with
variance <= 12 * 0.01
ACTION OF RULE #27
Trail typed as SINUSOIDAL OVERDENSE.

*

CONDITIONS OF RULE #28
[1] Amplitude range over trail is greater than 6 dBm.
[2] Trail duration greater than 5 times 50 mSec.
[3] A parabola of variance <= 12 * 0.1 covers
50% of the trail.
ACTION OF RULE #28
Trail typed as WIND-BLOWN OVERDENSE.

*

*

CONDITIONS OF RULE #29

[1] Variance from straight line on fall > 10 * 0.1.
[2] Trail duration less than or equal to 40 times 50 mSec.
[3] Straight line variance over trail > 2 times
variance over rise.
[4] There are less than 5 local extrema.
[5] Variance from straight line on rise <= 10 * 0.1.
[6] Line segments needed over fall with tolerance 6 is > 1.
[7] Line segments calculated above rise for <= 10 % of fall.
[8] Segments calculated above with
slope < ABS(10 * 0.01) is > 0.

ACTION OF RULE #29

Trail typed as MULTI-PLATEAU UNDERDENSE.

*

CONDITIONS OF RULE #30

[1] Variance from straight line on fall > 10 * 0.1.
[2] Trail duration less than or equal to 40 times 50 mSec.
[3] Straight line variance over trail > 2 times
variance over rise.
[4] There are less than 5 local extrema.
[5] Variance from straight line on rise <= 10 * 0.1.
[6] Line segments needed over fall with tolerance 6 is > 1.
[7] Line segments calculated above rise for <= 10 % of fall.
[8] Segments calculated above with
slope < ABS(10 * 0.01) <= 0.

ACTION OF RULE #30

Trail typed as MULTI-SLOPE UNDERDENSE.

*

CONDITIONS OF RULE #31

[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st third of trail.
[3] Straight line variance over trail > 2 times
variance over rise.
[4] Amplitude range over trail is greater than 3 dBm.
[5] Straight line variance is greater than 8 times 0.1.
[6] Good parabola fit from most pronounced
local min. on fall to end of trail.
[7] Straight line variance over trail > 1 times
variance over fall.
[8] Upper plateau comprises less or equal to 25 % of trail.
[9] Variance from straight line on fall > 8 * 0.1.

ACTION OF RULE #31

Trail typed as HUMP-BACKED CLASSIC.

*

CONDITIONS OF RULE #32

[1] Trail duration less than or equal to 40 times 50 mSec.
[2] Peak is in 1st third of trail.
[3] Straight line variance over trail > 3 times
variance over fall.
[4] Amplitude range over trail is greater than 3 dBm.
[5] Straight line variance is greater than 5 times 0.1.
[6] Variance from straight line on rise <= 30 * 0.1.
[7] Variance from straight line on rise > 10 * 0.1.
[8] Upper plateau comprises less or equal to 15 % of trail.
[9] Variance from straight line on fall <= 10 * 0.1.

ACTION OF RULE #32

Trail typed as CLASSIC UNDERDENSE WITH BAD ("Drunk") RISE.

Appendix D - Throughput Simulation Results

Bit Rate	Block Size	Blocks Tried	Block % OK	Kbits Sent
143750	4574	10134	54.16	25062
143750	5573	9792	48.14	26231
143750	6572	9155	44.80	26921
143750	7571	7939	41.97	25201
151250	4574	11612	51.43	**27267**
151250	5573	9246	48.56	24982
151250	6572	9344	41.96	25731
151250	7571	8709	39.57	26059
158750	4574	11768	48.54	26084
158750	5573	10713	44.80	26709
158750	6572	8790	42.52	24530
158750	7571	8220	40.12	24939
166250	4574	9003	60.71	24958
166250	5573	8613	53.63	25705
166250	6572	8266	48.16	26132
166250	7571	7645	45.43	26264

Datacom Optimisation Process (1100km link) - Iteration 3

Appendix E - Statistical Derivation

Having a mean μ and a standard deviation σ of delay associated with each link, the simulator required that random numbers be generated over an interval in such a way as to ensure that the numbers generated had the same mean and standard deviation as were associated with the link. The derivation of this interval is described here.

Now a random distribution over an interval will have the same mean and standard deviation as a uniform distribution, given that we have an infinite number of samples. Clearly then the centre of the interval must be the mean associated with the link, and we seek to determine k such that an infinite number of random numbers generated in the range $(\mu - k\,\sigma, \mu + k\,\sigma)$ will have a mean equal to μ and a standard deviation of σ. As we have a uniform distribution, we may consider the equivalent range of $(0, 2\,k\,\sigma)$ and a corresponding mean of $k\,\sigma$.

We now use a discrete uniform distribution with n+1 points to derive k, and then determine the result as n tends to infinity. We shall work with variance rather than standard deviation for ease of computation.

The n + 1 points occurring uniformly across the range 0 to $2\,k\,\sigma$ are

$$0,\ 1.\frac{2\,k\,\sigma}{n},\ 2.\frac{2\,k\,\sigma}{n},\ 3.\frac{2\,k\,\sigma}{n},\ \ldots,\ n.\frac{2\,k\,\sigma}{n}$$

Their mean, $k\,\sigma$, can be written as $\frac{n}{2}.\frac{2\,k\,\sigma}{n}$, giving us the following formula for their variance :

$$\frac{1}{n+1}\sum_{i=0}^{n}\left(\frac{2k\sigma}{n}\left(i-\frac{n}{2}\right)\right)^2$$

$$= \frac{1}{n+1}\sum_{i=0}^{n}\left(\frac{4k^2\sigma^2}{n^2}\left(i^2 - ni + \frac{n^2}{4}\right)\right)$$

$$= \frac{1}{n+1}\left(\frac{4k^2\sigma^2}{n^2}\frac{n^2}{4} + \sum_{i=1}^{n}\left(\frac{4k^2\sigma^2}{n^2}\left((i^2 - ni + \frac{n^2}{4})\right)\right)\right)$$

$$= \frac{1}{n+1}\left(k^2\sigma^2 + \frac{4k^2\sigma^2}{n^2}\left[\frac{n(n+1)(2n+1)}{6} - n\frac{n(n+1)}{2} + \frac{n^3}{4}\right]\right)$$

$$= \frac{1}{n+1}\left(k^2\sigma^2 + \frac{4k^2\sigma^2}{n^2}\left[\frac{4n^3 + 6n^2 + 2n}{12} - \frac{6n^3 + 6n^2}{12} + \frac{3n^3}{12}\right]\right)$$

$$= \frac{1}{n+1}\left(k^2\sigma^2 + \frac{4k^2\sigma^2}{n^2}\left[\frac{n^3 + 2n}{12}\right]\right)$$

$$= \frac{1}{n+1}\left[k^2\sigma^2 + \frac{k^2\sigma^2(n^2+2)}{3n}\right]$$

$$= k^2\sigma^2\left(\frac{1}{n+1}\left[1 + \frac{n^2+2}{3n}\right]\right)$$

Let this equation equal σ^2

then $k^2\left(\frac{1}{n+1}\left[1 + \frac{n^2+2}{3n}\right]\right) = 1$

$$\Rightarrow k^2\left(\frac{1}{n+1}\left[\frac{n^2+3n+2}{3n}\right]\right) = 1$$

$$\Rightarrow k^2\left(\frac{1}{n+1}\left[\frac{(n+1)(n+2)}{3n}\right]\right) = 1$$

$$\Rightarrow k^2\,\frac{(n+2)}{3n} = 1$$

$$\Rightarrow k^2 = \frac{3n}{(n+2)} = 3\left(\frac{n}{n+2}\right)$$

Now, $\lim_{n\to\infty} \frac{n}{n+2} = 1$

so $\lim_{n\to\infty} k^2 = 3$

$\Rightarrow k = \sqrt{3}$ ❑

Appendix F - Numerical Results of Simulation

Error rate = **0.05**

Routing strategy	Mean Delay	Delay Std Deviation
Minimum-Hop	75.82	56.74
Shortest-Path	63.98	43.11
Flooding	50.30	34.56

Error rate = **0.1**

Routing strategy	Mean Delay	Delay Std Deviation
Minimum-Hop	76.11	57.12
Shortest-Path	64.49	49.38
Flooding	51.50	34.70

Error rate = **0.2**

Routing strategy	Mean Delay	Delay Std Deviation
Minimum-Hop	99.83	69.53
Shortest-Path	81.79	62.80
Flooding	59.22	34.85

Error rate = **0.5**

Routing strategy	Mean Delay	Delay Std Deviation
Minimum-Hop	134.05	121.94
Shortest-Path	125.78	112.09
Flooding	74.58	54.87

Error rate = **0.8**

Routing strategy	Mean Delay	Delay Std Deviation
Minimum-Hop	344.23	323.26
Shortest-Path	256.63	219.20
Flooding	144.72	124.99

Appendix G - Connectivity Results

Connectivity	Minimum-Hop	Flood
2	440.12	377.44
4	220.66	130.78
6	161.26	62.21
8	124.45	32.17
10	103.22	18.99
12	89.31	11.58
14	85.76	6.67
16	74.75	4.21
18	70.23	2.76
20	62.64	1.80
22	62.72	1.14
24	62.60	0.79
26	56.33	0.50
28	53.64	0.37
30	51.79	0.23
32	50.54	0.19
34	48.01	0.13
36	45.73	0.13
38	40.68	0.08
39	40.61	0.08

Appendix H - Source Listing of the TrailStar System

```
PROGRAM Trailstar;

{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
{~~~                    TrailStar                              ~~~}
{~~~                                                           ~~~}
{~~~    Version 3.1            Last Modified : July, 1991      ~~~}
{~~~                                                           ~~~}
{~~~     Expert System Design by :  Stuart Melville            ~~~}
{~~~                                                           ~~~}
{~~~     Programming by :           Stuart Melville,           ~~~}
{~~~                              Robert Letschert             ~~~}
{~~~                                                           ~~~}
{~~~     Initial Domain Expert :    James Larsen               ~~~}
{~~~                                                           ~~~}
{~~~     Mathematical Consultant :  Wayne Goddard              ~~~}
{~~~                                                           ~~~}
{~~~    This system can classify, and learn to classify,       ~~~}
{~~~    meteor trail reflections.                              ~~~}
{~~~                                                           ~~~}
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}

{$R-}    {Range checking off}
{$B+}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$N-}    {No numeric coprocessor}
{$I-}    {Interrupt checking off}
{$V-}


USES
  Crt, Dos, Printer, Graph,
  {$U TS1. } Ts1, {$U TS2. } Ts2, {$U TS3. } Ts3, {$U TS4. } Ts4,
  {$U TS5. } Ts5, {$U TS6. } Ts6, {$U TS7. } Ts7, {$U TS8. } Ts8,
  {$U TS9. } Ts9, {$U TS10. } Ts10, {$U TS11. } Ts11,
  {$U TS12. } Ts12, {$U TS13. } Ts13;



{____________________________________________________________________}
PROCEDURE Show_Menu_And_Get_Choice;
{Displays main menu and gets user's choice.}
   CONST Esc = #27;

BEGIN
    Menu_Head;
    WRITE('         USER-TYPING  MENU');
    Gotoxy( 45, 3);
    WRITE('Current file name : ', Curname);
    Gotoxy( 45, 4);
    WRITE('Current path : ',Path);
    Gotoxy( 45, 5);
    WRITE('Current rule base : ', Curbase);
    WRITELN;
    Menu_Write('T =  Type a Trail.');
    Menu_Write('B =  change current rule Base');
    Menu_Write('C =  Change current trail file.');
    Menu_Write('D =  Determine base used for typing trails');
    Menu_Write('I =  enter Instruction mode.');
    Menu_Write('L =  enter Listing module.');
    Menu_Write('P =  change Path');
    Menu_Write('H =  get Help.');
    Menu_Write('X =  eXit.');
    Gotoxy(1,24);
    WRITE('Please enter choice : ');
    Choice:= Get_Valid_Choice_From([Esc,'P','B','C','D','L','I','H','T','X'])
END;
{____________________________________________________________________}
```

```
{_____________________________________________________________________}
FUNCTION Get_File_From_Table : String;
   VAR Head_File_Num : INTEGER;

BEGIN
   List_File_Table_To_Screen;
   Head_File_Num := Get_User_Int ('Enter file number :',
                                  1, Number_Files_In_Table);
   IF Exitting THEN Exit;
   Get_File_From_Table:= File_Table [Head_File_Num];
END;
{_____________________________________________________________________}


{_____________________________________________________________________}
FUNCTION Get_File_From_User : String;
    VAR Hnam, Snam : String;
           Old_Dir : String;
              Badf : BOOLEAN;

BEGIN
  Getdir( 0, Old_Dir);
  {$I-}
  REPEAT
     Clear_From( 23);
     Gotoxy( 1, 24);
     Clreol;
     Gotoxy(1,24);
     WRITE('Please enter new trail file name : ');
     READ( Hnam);
     Chdir( Path);
     ASSIGN( Trail_Head, Hnam);
     RESET( Trail_Head);
     Badf:= (Ioresult <> 0);
     Snam:= Hnam;
     Snam[1]:= 'S';
     ASSIGN( Trail_Sample, Snam);
     RESET( Trail_Sample);
  UNTIL ((NOT Badf) AND (Ioresult = 0));
  {$I+}
  CLOSE( Trail_Head);
  CLOSE( Trail_Sample);
  Chdir( Old_Dir);
  Get_File_From_User:= Hnam;
END;
{_____________________________________________________________________}
```

```
{_____________________________________________________________________}
PROCEDURE Change_Path;
  VAR Oldpath : String;
      Old_Dir : String;
       Ok_Dir : BOOLEAN;

BEGIN
   Menu_Head;
   WRITELN('                    PATH SELECTION');
   WRITELN;
   Oldpath:= Path;
   Getdir( 0, Old_Dir);
   Ok_Dir:= FALSE;
   WRITELN('If want current directory as path then just hit return.');
   REPEAT
       WRITE('Please enter new path name (eg. A:\DATA\METEOR) : ');
       READLN( Path);
       {$I-}
         Chdir( Path);
       {$I+}
       IF (Ioresult <> 0) THEN BEGIN
           Clrscr;
           Gotoxy(1,11);
           WRITE('Specified path ', Path,' not found! ');
           WRITE('Press any key to retry...');
           Anykey:= Readkey;
           Path:= 'UNNOWN';
        END
        ELSE Ok_Dir:= TRUE;
        Chdir( Old_Dir);
   UNTIL Ok_Dir;
   IF (Oldpath <> Path) THEN BEGIN
       {$I-}          {Close if open.}
       CLOSE( Trail_Head);
       CLOSE( Trail_Sample);
       {$I+}
       IF (Ioresult = 0) THEN BEGIN {do nothing.} END;
       Curname:= 'UNKNOWN';
   END;
   Clrscr;
END;
{_____________________________________________________________________}


{_____________________________________________________________________}
PROCEDURE Handle_Type_Trail_Call;

BEGIN
   IF (Curname <> 'UNKNOWN') THEN
        Type_Trail
   ELSE BEGIN
        Gotoxy(1,24);
        Clreol;
        Gotoxy(1,23);
        WRITE('Must specify a trail file first - use "C" option.');
        Pause_It;
        Clear_From( 23)
    END
END;
{_____________________________________________________________________}
```

```
{_____________________________________________________________________}
PROCEDURE Determine_Base;

BEGIN
  Clear_From( 23);
  IF (Curname <> 'UNKNOWN') THEN BEGIN
        Gotoxy(1,23);
        WRITE('Base used was ');
        IF (Base_Used <= 100) THEN
            WRITE( Base_Used)
        ELSE WRITE( 'unknown - trail typed by old program.');
  END
  ELSE BEGIN
       Gotoxy(1,23);
       WRITE('You must specify a trail file first! Use ''C'' option.');
  END;
  Pause_It;
  Clear_From( 23);
END;
{_____________________________________________________________________}


{_____________________________________________________________________}
PROCEDURE Enter_User_Typing_Mode;
{Checks if the high-level modifying suite of routines wanted, in}
{which case passes control to these, otherwise  repeatedly shows}
{main menu and passes control to routine appropriate  to  user's}
{choice until user chooses to exit from the BridgeStar package. }
   CONST Esc = #27;

BEGIN
   Curname:= 'UNKNOWN';
   Path:= '';
   Auto_Type := FALSE;
   REPEAT
      Show_Menu_And_Get_Choice;
      CASE Choice OF
          Esc : High_Level_Modifier;
          'L' : Lister;
          'T' : Handle_Type_Trail_Call;
          'C' : Change_Curname( Base_Used);
          'B' : Change_Base;
          'D' : Determine_Base;
          'H' : Do_Help;
          'P' : Change_Path;
          'I' : Learn;
          'X' : {Do nothing}
      END;
   UNTIL ((Choice = 'X') OR (Choice = Esc))
END;
{_____________________________________________________________________}
```

```
{_____________________________________________________________}
PROCEDURE Auto_Type_Files_In_Current_Directory;

VAR  File_Number, Start_File, Stop_File    : INTEGER;
                              Sel_Choice   : CHAR;
                                  Daystr   : String;

   {_____________________________________________}
   PROCEDURE Give_Selection_Options;

   BEGIN
     Menu_Head;
     WRITELN('       FILE SELECTION OPTIONS');
     WRITELN;
     Menu_Write('A = auto-type All files, regardless of any previous typing.');
     Menu_Write('T = auto-Type all previously untyped files.');
     Menu_Write('M = Manually select files to auto-type.');
     Menu_Write('D = auto-type all files in a selected Day.');
     Menu_Write('S = select a consecutive String of files to auto-type.');
     Menu_Write('X = eXit. (Can be done mid-typing by using Esc key)');
     WRITELN;
     WRITELN;
     WRITE('Please enter choice : ');
     Sel_Choice:= Get_Valid_Choice_From(['A','D','M','S','T','X']);
     Clrscr;
   END;
    {_____________________________________________}


   {_____________________________________________}
   PROCEDURE Get_String_Wanted;

   BEGIN
     List_File_Table_To_Screen;
     Start_File:= Get_User_Int('Start file number? ',1, Number_Files_In_Table);
     IF Exitting THEN Exit;
     Stop_File:= Get_User_Int('Stop file number? ',1, Number_Files_In_Table);
     IF Exitting THEN Exit;
   END;
   {_____________________________________________}


   {_____________________________________________}
   PROCEDURE Adjust_File_Table_On_Choices;
     VAR Table_Pos, File_Possible, Fnum : INTEGER;
                          Fnums_Chosen : ARRAY[1..1000] OF BOOLEAN;

   BEGIN
      Fillchar( Fnums_Chosen, Sizeof( Fnums_Chosen), FALSE);
      Menu_Head;
      WRITELN('You will now be given a list of the various files available');
      WRITELN('- make a note of the numbers of those files you wish typed.');
      WRITE('Press any key to begin display...');
      Pause;
      List_File_Table_To_Screen;
      Gotoxy( 1, 24);
      REPEAT
        Fnum:= Get_User_Int( 'Enter file numbers wanted (0 to quit entry) :',
                             0, Number_Files_In_Table);
        IF Exitting THEN Exit;
        IF (Fnum <> 0) THEN
            Fnums_Chosen[ Fnum]:= TRUE;
      UNTIL (Fnum = 0);
      Table_Pos:= 0;
      FOR File_Possible:= 1 TO Number_Files_In_Table DO
          IF (Fnums_Chosen[ File_Possible] = TRUE) THEN BEGIN
                Table_Pos:= Table_Pos + 1;
                File_Table[ Table_Pos]:= File_Table[ File_Possible];
          END;
      Start_File:= 1;
      Stop_File:= Table_Pos;
   END;
   {_____________________________________________}
```

```
{_____________________________________________________}
PROCEDURE Get_The_Day_Wanted;
  VAR Dag : INTEGER;
      Sys : CHAR;

BEGIN
   Menu_Head;
   WRITELN('     DAY AND SYSTEM SELECTION');
   WRITELN;
   WRITE('Please enter system (A -> Z) : ');
   Sys:= Get_Valid_Choice_From( ['A'..'Z']);
   WRITELN;
   Dag:= Get_User_Int('Please enter day (1 -> 365) : ', 1, 365);
   STR( Dag, Daystr);
   IF (Dag < 10) THEN
        Daystr:= '0' + Daystr;
   IF (Dag < 100) THEN
        Daystr:= '0' + Daystr;
   Daystr:= 'HD' + Sys + Daystr;
END;
{_____________________________________________________}


BEGIN
  Auto_Type := TRUE;
  Change_Path;
  Clrscr;
  Fill_File_Table;
  Give_Selection_Options;
  CASE Sel_Choice OF
     'A', 'D', 'T' : BEGIN   {The actual routine differentiates between them.}
                        Start_File:= 1;
                        Stop_File:= Number_Files_In_Table;
                     END;
                'M' : Adjust_File_Table_On_Choices;
                'S' : Get_String_Wanted;
                'X' : Exit;
  END;
  IF (Sel_Choice = 'D') THEN
      Get_The_Day_Wanted;
  IF Exitting THEN BEGIN
      Exitting:= FALSE;
      Exit;
  END;
  Clrscr;
  First_File_To_Auto_Type := TRUE;
  FOR File_Number := Start_File TO Stop_File DO
     IF Exitting THEN BEGIN
          Exitting:= FALSE;
          Exit;
      END
      ELSE Auto_Type_File (File_Table [File_Number], Sel_Choice, Daystr)
END;
{_____________________________________________________}


{_____________________________________________________}
PROCEDURE Scan_Through_File_Checking_Types;

VAR  Header_File, Sample_File           : Str12;
     Number_Trail_Types                 : INTEGER;
     Continue, Successful, Stay_In_File : BOOLEAN;
     Resp                               : Str8;
     The_Type, Trail_Type               : BYTE;
     New_Action                         : Condition;
     Header_Rec                         : Trail_Header;
     Start_Trail, Stop_Trail, Start_Pos : INTEGER;
     Answer                             : Str8;
     Start_Trail_Str, Stop_Trail_Str    : Str12;
     Sillykey                           : Str8;
```

```
{_____________________________________________}
FUNCTION Init_Scan_Ok : BOOLEAN;
   VAR Schoice : CHAR;
       Old_Dir : String;

BEGIN
  Change_Path;
  Risepos := 1;
  Fallpos := 1;
  Fill_File_Table;
  Clrscr;
  Gotoxy( 10,10);
  WRITE('Select from file Table or Directly specify file (T/D) ? ');
  Schoice:= Get_Valid_Choice_From([ Esc, 'T', 'D']);
  IF (Schoice = Esc) THEN BEGIN
       Init_Scan_Ok:= FALSE;
       Exitting:= TRUE;
       Exit;
  END;
  IF (Schoice = 'T') THEN
       Header_File:= Get_File_From_Table
  ELSE Header_File:= Get_File_From_User;
  IF Exitting THEN Exit;
  Clrscr;
  Menu_Head;
  Sample_File := Header_File;
  Sample_File [1] := 'S';
  Getdir( 0, Old_Dir);
  Chdir( Path);
  ASSIGN (Trail_Head, Header_File);
  RESET (Trail_Head);
  Hnam := Header_File;
  Snam := Sample_File;
  ASSIGN (Trail_Sample, Sample_File);
  {$I-}  RESET (Trail_Sample);  {$I+}
  Init_Scan_Ok:= (Ioresult = 0);
  Chdir( Old_Dir);
END;
{_____________________________________________}


{_____________________________________________}
PROCEDURE Get_Type_Alteration;

BEGIN
    Gotoxy(1,24);
    WRITE('Please enter new typing : ');
    READLN(The_Type);
    New_Action.Cond_Num:= The_Type;
    Determine_Action( New_Action, The_Type);
END;
{_____________________________________________}


{_____________________________________________}
PROCEDURE Set_Up_File;

BEGIN
   SEEK( Trail_Head, 0);
   READ (Trail_Head, Header_Rec);
   Start_Trail := Header_Rec.Trail_Num;
   SEEK (Trail_Head, Filesize (Trail_Head) - 1);
   READ (Trail_Head, Header_Rec);
   Stop_Trail := Header_Rec.Trail_Num;
   SEEK( Trail_Head, 0);
END;
{_____________________________________________}
```

```
BEGIN
  IF (NOT Init_Scan_Ok) THEN BEGIN
     IF Exitting THEN BEGIN
          Exitting:= FALSE;
          Exit
     END
     ELSE
       Sillykey:= Users_Response_To (1, 'File ' + Sample_File +
                    ' not on current drive (OK)?')
  END
  ELSE BEGIN
     Stay_In_File := TRUE;
     WHILE Stay_In_File DO BEGIN
        Set_Up_File;
        REPEAT
          Answer := Users_Response_To (1,
             'Start at beginning of file (Y/N)?');
        UNTIL ((Answer = 'Y') OR (Answer = 'N'));
        IF Answer = 'N' THEN BEGIN
           STR (Start_Trail, Start_Trail_Str);
           STR (Stop_Trail, Stop_Trail_Str);
           Start_Pos := Get_User_Int ('Enter trail between ' + Start_Trail_Str
              + ' and ' + Stop_Trail_Str + ' :  ', Start_Trail, Stop_Trail);
           IF Exitting THEN BEGIN
              Exitting:= FALSE;
              Exit;
           END;
           Header_Rec.Trail_Num := -1;
           WHILE (NOT (EOF (Trail_Head))) AND
                 (Header_Rec.Trail_Num <> Start_Pos) DO
              READ (Trail_Head, Header_Rec);
           SEEK (Trail_Head, Filepos (Trail_Head) - 1);
        END;
        Get_Trail_Type_From_User (Trail_Type);
        IF Exitting THEN BEGIN
            Exitting:= FALSE;
            Exit;
        END;
        Continue := TRUE;
        WHILE (NOT (EOF (Trail_Head))) AND (Continue) DO BEGIN
          Get_Next_Trail_Type (Trail_Type, Successful);
          IF Successful THEN BEGIN
             Interpolate_Trail;
             Display_Trail( Scan_Call);
             Menu_Head;
             Resp := Users_Response_To (1,
                'Continue (Y/N)? or A to Alter typing');
             IF (Resp = 'A') THEN
                Get_Type_alteration;
             Continue:= ((Resp = 'A') OR (Resp = 'Y'))
          END;
        END;
        IF EOF (Trail_Head) THEN BEGIN
           Menu_Head;
           Sillykey := Users_Response_To (1, 'No more trails of specified ' +
                                             'type left in file (OK)?')
        END;
        Stay_In_File := Users_Response_To
           (1, 'Scan another type in this file (Y/N)') = 'Y'
     END;
     CLOSE (Trail_Sample)
  END;
  CLOSE (Trail_Head)
END;
{_____________________________________________________________}
```

```
{_____________________________________________________________________}
FUNCTION Get_Base_Number : INTEGER;
  VAR Bnum : INTEGER;

BEGIN
  Menu_Head;
  WRITELN('                RULE  BASE  CREATION');
  WRITELN('                ~~~~~~~~~~~~~~~~~~~~');
  WRITELN('This will create a new rule base, with just two default');
  WRITELN('rule stored inside it (an unreasonable data test  rule,');
  WRITELN('and the  assignation to  the unknown type used when all');
  WRITELN('else fails). Be sure to always keep a unreasonable data');
  WRITELN('test rule in highest priority, and a default assignment');
  WRITELN('in the lowest priority position! After this processing,');
  WRITELN('the  Learn module (in  User typing menu) can be used to');
  WRITELN('add further rules.');
  WRITELN;
  WRITE('Please enter number of the new rule base : ');
  Bnum:= Get_Good_Int( 1, 100);
  IF (Bnum IN Bases_Available) THEN BEGIN
      WRITELN;
      WRITELN('There is already a rule base with this number. This will be');
      WRITELN('destroyed if you continue!!! Want to continue (Y/N) ? ');
      IF ( Upcase( Readkey) = 'N') THEN
          Bnum:= 0;
  END;
  Get_Base_Number:= Bnum;
END;
{_____________________________________________________________________}


{_____________________________________________________________________}
PROCEDURE Create_Base;
  VAR Initial_Rule : Rules;
            R_Pri : INTEGER;

BEGIN
   Basenum:= Get_Base_Number;
   IF (Basenum <> 0) THEN BEGIN
      STR( Basenum, Curbase);
      Bases_Available:= Bases_Available + [Basenum];
      ASSIGN( Rbase, 'RULES' + Curbase + '.STA');
      REWRITE( Rbase);
      Initial_Rule[ 0].Cond_Num:= 1;  {Number of conditions.}
      Initial_Rule[ 1].Cond_Num:= 25; {Unreasonable data test.}
      Initial_Rule[20].Cond_Num:= 1;  {Unreasonable data typing.}
      WRITE( Rbase, Initial_Rule);
      Initial_Rule[ 0].Cond_Num:= 1;  {Number of conditions.}
      Initial_Rule[ 1].Cond_Num:= 1;  {Indicates dummy condition, always true.}
      Initial_Rule[20].Cond_Num:= 22; {Gothic rocker typing.}
      WRITE( Rbase, Initial_Rule);
      CLOSE( Rbase);
      ASSIGN( Pribase, 'PRIOR' + Curbase + '.STA');
      REWRITE( Pribase);
      FOR R_Pri:= 1 TO 2 DO
         WRITE( Pribase, R_Pri);
      CLOSE( Pribase);
      ASSIGN( Rbase, 'RULES' + Curbase + '.STA');
      RESET( Rbase);
      ASSIGN( Pribase, 'PRIOR' + Curbase + '.STA');
      RESET( Pribase);
   END;
END;
{_____________________________________________________________________}
```

```
{_____________________________________________________________________}
PROCEDURE Show_Main_Menu_And_Get_Choice;
{Displays main menu and gets user's choice.}

BEGIN
    Clrscr;
    Menu_Head;
    WRITELN;
    WRITELN;
    WRITE('           MAIN  MENU');
    Lowvideo;
    WRITELN;
    WRITELN;
    Menu_Write('A =  Auto-type file(s) in current Directory.');
    Menu_Write('B =  Change rule base used. (Current base : ' + Curbase + ')');
    Menu_Write('C =  Create a new rule base.');
    Menu_Write('S =  Scan through a file checking types.');
    Menu_Write('U =  enter User typing mode.');
    Menu_Write('X =  eXit.');
    Gotoxy(1,24);
    WRITE('Please enter choice : ');
    Choice:= Get_Valid_Choice_From (['A', 'B', 'C', 'S', 'U', 'X'])
END;
{_____________________________________________________________________}



{_____________________________________________________________________}
PROCEDURE Process;
{Checks if the high-level modifying suite of routines wanted, in}
{which case passes control to these, otherwise  repeatedly shows}
{main menu and passes control to routine appropriate  to  user's}
{choice until user chooses to exit from the BridgeStar package. }

VAR  Exit    : BOOLEAN;

BEGIN
   Exit := FALSE;
   REPEAT
      Show_Main_Menu_And_Get_Choice;
      IF (Choice IN ['U','A']) AND (Basenum = 0) THEN
           Change_Base;   {Need a rule base to type with!}
      CASE Choice OF
          'U' : Enter_User_Typing_Mode;
          'S' : Scan_Through_File_Checking_Types;
          'B' : Change_Base;
          'C' : Create_Base;
          'A' : Auto_Type_Files_In_Current_Directory;
          'X' : Exit := TRUE;
      END;
   UNTIL Exit;
END;
{_____________________________________________________________________}


{Main Program}

BEGIN
  Initialise;
  IF Files_Ok THEN
     Process;
  Finalise
END.
```

```
UNIT Ts1;
{This unit contains global constant, type and variable definitions,}
{as well as utility routines.                                      }

Interface

USES Dos, Crt;

CONST
              Priority_Order = FALSE;
                Number_Order = TRUE;

                Leave_Cursor = FALSE;
                 Move_Cursor = TRUE;

                 Typing_Call = TRUE;
                   Scan_Call = FALSE;

                Wait_Between = TRUE;
                     Play_On = FALSE;

                    Computer = TRUE;
                        User = FALSE;

                       Minim = 1;
                        Maxi = 0;

                        High = FALSE;
                         Low = TRUE;

                    Topbase = 100;

           Max_Failed_Rules = 200;

                  Max_Files = 1000;
                Max_Samples = 10000;

               Cubic_Cutoff = 150;
               Two_Polly_Max = 10;
                   Polly_max = 5;
                    Db_Scale = 4;

                         Esc = #27;

{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
TYPE  Complex = RECORD
         r     : REAL;
         i     : REAL
      END;

      Complex_Array_Struct = ARRAY [1..2048] OF Complex;
      Complex_Array        = ^complex_Array_Struct;


      Line_Node = RECORD
         Start, Stop    : INTEGER;
         B0, B1         : REAL
      END;


      Unsigned = ARRAY[1..2] OF BYTE;

      Db_Over_Noise = (Plus6, Plus9, Plus12, Plus15);


      Time_Rec = RECORD
          Hours, Mins, Secs, Huns : BYTE
      END;
```

```
      Trail_Header = RECORD                    {Main trail record.}
                       Trail_Num : INTEGER;
                       Start_Time : Time_Rec;
                       Elapsed_Time : REAL;
                       Num_Samples : INTEGER;
                       Sample_Range : ARRAY [0..1] OF REAL;
                       Background_Noise, Number_Of_Fades : INTEGER;
                       Peak_Time, Peak_Strength : INTEGER;
                       Signal_Present_Time : ARRAY[0..1] OF Unsigned;
                       Useful_Time : ARRAY[ Plus6..Plus15] OF Unsigned;
                       Trail_Type : BYTE;
                       Sample_Mean : REAL;
                       Wait_Time_From_Last : REAL;
                       Errors_Found : BYTE
                     END;


      Xyvals = (Xval, Yval);
      Seplatts = (Splatt, Eplatt);     {Plateau begins and ends.}

      Data_Array     = ARRAY [0..Max_Samples] OF INTEGER;
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
      Str2 = String[2];
      Str5 = String[5];
      Str8 = String[8];
      Str12 = String[12];
      Str20 = String[20];
      Str40 = String[40];
      Str80 = String[80];

      Charset = SET OF CHAR;

      Bases = SET OF 1..100;


        Intrpt  = RECORD
                    Ax, Bx, Cx, Dx, Dp, Di, Si, Ds, Es, Fl : INTEGER
                  END;

      Condition = RECORD
                      Cond_Num : BYTE;
                      C_Parms  : ARRAY[1..3] OF BYTE
                  END;

       Rule_And_Condition = RECORD
                              Rnumber : INTEGER;
                              Cnumber : BYTE;
                            END;

       Rules = ARRAY[0..20] OF Condition;
       Str80file = FILE OF Str80;
       Rulefile = FILE OF Rules;
       Int_File = FILE OF INTEGER;
       Bytefile = FILE OF BYTE;
```

```
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
VAR
       Graphdriver, Graphmode, Errorcode : INTEGER;
     Num_Extrema, Num_Minima, Num_Maxima : BYTE;
                 Trail_Array, Disp_Array : Data_Array;
                         Minima, Maxima : ARRAY[1..50,Xval..Yval] OF INTEGER;
                      debug_flag, Quit  : BOOLEAN;
                    Length_Up_Plateau   : INTEGER;
                           Plat_Count   : INTEGER;
                             Plateaus   : ARRAY[1..100, Splatt..Eplatt]
                                          OF INTEGER;
                           Hnam, Snam   : String [14];
                       Hrec, Prev_Rec   : Trail_Header;
                 Start_Trail, End_Trail : INTEGER;
                             Fourrier   : Complex_Array;
                 Power, Fourrier_Points : INTEGER;
                               Coeff    : ARRAY [1..5] OF REAL;
                               Param    : REAL;
                          File_Table    : ARRAY [1..1000] OF Str12;
                Number_Files_In_Table   : INTEGER;
           A_Coeff, B_Coeff, C_Coeff    : REAL;
                           Parab_Var    : REAL;

        X_Bar, Y_Bar, B0, B1, Variance  : REAL;
                  Down_X_Bar, Down_Y_Bar : REAL;
       Down_B0, Down_B1, Down_Variance  : REAL;
                    Up_X_Bar, Up_Y_Bar  : REAL;
          Up_B0, Up_B1, Up_Variance     : REAL;
       Part_B0, Part_B1, Part_Variance  : REAL;
                  Part_X_Bar, Part_Y_Bar : REAL;


     High_Tide, Low_Tide, Highest_Platt : INTEGER;
                    Peak_Pos, Min_Pos   : INTEGER;
                      Fallpos, Risepos  : INTEGER;
                           Fade_Count   : INTEGER;
                            Top_Total   : INTEGER;
                         Prob_Plateau   : BOOLEAN;
                            Peak_Pos2   : INTEGER;
                         Counts_Array   : ARRAY[-135..-80] OF INTEGER;
                     Loc_Max, Loc_Min   : ARRAY[1..1000] OF INTEGER;
                   Num_Maxes, Num_Mins  : INTEGER;
                         Garbage_Flag   : BOOLEAN;
                                  Tau   : REAL;
                    Number_Of_Lines     : INTEGER;
                                Lines   : ARRAY [1..50] OF Line_Node;
                           Auto_Type    : BOOLEAN;
                Total_Result, Maxlowdif : REAL;
                    Worst_Line_Dif_Pos  : INTEGER;
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
                          Abase, Cbase  : Str80file;
                               Pribase  : Int_File;
                                 Rbase  : Rulefile;

                           Trail_Head   : FILE OF Trail_Header;
                         Trail_Sample   : Bytefile;

                          Yesno, Spaces : Charset;
     sound_flag, printer_handy, Files_Ok : BOOLEAN;
                First_File_To_Auto_Type  : BOOLEAN;
                                Curname  : Str20;

                    Before_Next_Choice   : INTEGER;
                         Test_Failures   : INTEGER;
                          First_Choice   : INTEGER;
                           First_Fired   : Rules;
                            Last_Fired   : Rules;
                               Failures  : ARRAY[ 1..Max_Failed_Rules] OF
                                             Rule_And_Condition;
                      Bases_Available   : Bases;
                            Base_Used   : BYTE;
                              Basenum   : INTEGER;
                              Curbase   : String;
```

```
        User_Wants_Actual_Condition_Value : BOOLEAN;

              Choice, Play_Choice, Anykey : CHAR;
                                  Exitting : BOOLEAN;
                                      Path : String;
                                       Prt : TEXT;
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
PROCEDURE Menu_Head;
PROCEDURE Pause;
PROCEDURE Fill_File_Table;
PROCEDURE List_File_Table_To_Screen;
FUNCTION Users_Response_To (Chars  : INTEGER; Question  : Str80) :Str8;
FUNCTION Get_Good_Int( Lowb, Upb : INTEGER) : INTEGER;
FUNCTION Get_User_Int (Question          : Str80;
                       Min_Int, Max_Int  : INTEGER)  : INTEGER;
PROCEDURE Get_Trail_Type_From_User (VAR Trail_Type   : BYTE);
FUNCTION Absolute_Val (Rval : REAL) : REAL;
PROCEDURE Debug( Outl : Str80);
PROCEDURE Seek_Rule( Rec_Num : BYTE; VAR Rule_Back : Rules);
PROCEDURE Seek_Cond( Rec_Num : BYTE; VAR Cond_Back : Str80);
PROCEDURE Seek_Act( Rec_Num : BYTE; VAR Act_Back : Str80);
FUNCTION Conds_In (Some_Rule : Rules) : BYTE;
FUNCTION Eo_St(St_In : Str80; Pos : BYTE) : BOOLEAN;
PROCEDURE Get_Item( VAR Item : Str80; VAR Pos : BYTE; St_In : Str80);
PROCEDURE Pause_It;
PROCEDURE Wait_Till_Read;
PROCEDURE Clear_From( Screen_Line : BYTE);
PROCEDURE Alarm( Move : BOOLEAN);
FUNCTION Get_Valid_Choice_From( Legalset : Charset) : CHAR;
PROCEDURE Help_Head;
PROCEDURE Menu_Write( St_In : Str80);
PROCEDURE Print_Ordinal( Parm_Num : BYTE);
PROCEDURE Print_Parm( Parm_Num : BYTE);
PROCEDURE Show_Cond( Cond_In : Condition);
PROCEDURE Write_Conds ( Rule_Containing : Rules);
PROCEDURE Show_Act( Act_In : Condition);
FUNCTION Float (INT  : INTEGER)  : REAL;
FUNCTION Round_Up (Reel  : REAL)  : INTEGER;
FUNCTION Number_Digits (Value  : INTEGER)  : BYTE;
PROCEDURE Kill_Lead_Zeroes( VAR Str_In : Str12);
PROCEDURE Get_Hour_File_Descrip( Name_Of_Hour_File : Str12; VAR Systm,
                                 Day_Of, Hour_Of, Year_Of : Str12);
PROCEDURE Fit_Lines_To_Trail (Start, Stop          : INTEGER;
                              Variance_Tolerance  : REAL;
                              Last_Line           : INTEGER);
PROCEDURE Stu_Parabola( Startp, Endp : INTEGER);
FUNCTION One_Time( Inrec : Time_Rec) : REAL;
FUNCTION Variance_From_Line( B_Zero, B_One : REAL;
                             Posa, Posn : INTEGER) : REAL;
PROCEDURE Linear_Regression_On_Part_Of_Trail( Startp, Endp : INTEGER);
PROCEDURE Change_Base;
PROCEDURE Show_Whynot_Act( Act_In : Condition);


IMPLEMENTATION
{$V-}


{*********************************************************************}
PROCEDURE Clear_From( Screen_Line : BYTE);
{Clears the screen from SCREEN_LINE down, and leaves cursor}
{at the beginning of SCREEN_LINE.                          }
   VAR I : BYTE;

BEGIN
   FOR I:= Screen_Line TO 25 DO BEGIN
       Gotoxy(1, I);
       Clreol
   END;
   Gotoxy(1, Screen_Line)
END;
{*********************************************************************}
```

```
{*********************************************************************}
PROCEDURE Menu_Head;
{Used to initialise the various menus.}

BEGIN
  Clrscr;
  Normvideo;
  WRITE('TrailStar                                          ');
  WRITELN('           Melville/Letschert (1987)');
  WRITELN
END;
{*********************************************************************}


{*********************************************************************}
PROCEDURE Pause;
  VAR Akey : CHAR;

BEGIN
  Akey:= Readkey;
END;
{*********************************************************************}


{*********************************************************************}
FUNCTION No_Samples_Or_Invalid_File( S_Fnam : Str12) : BOOLEAN;
   VAR Sfil_Checker : Bytefile;

BEGIN
   {First test that header name conforms to HD/HN + Sytem character.}
   IF ((S_Fnam[2] IN ['D','N']) AND (S_Fnam[3] IN ['A'..'Z'])) THEN BEGIN
     S_Fnam[ 1]:= 'S';  {Changed from 'H', referring to header.}
     {$I-}
        ASSIGN( Sfil_Checker, S_Fnam);
        RESET( Sfil_Checker);
     {$I+}
     IF (Ioresult <> 0) THEN
         No_Samples_Or_Invalid_File:= TRUE    {File doesn't even exist.}
     ELSE BEGIN
         No_Samples_Or_Invalid_File:= Filesize( Sfil_Checker) = 0;
         CLOSE( Sfil_Checker)
     END
   END
   ELSE No_Samples_Or_Invalid_File:= TRUE
END;
{*********************************************************************}


{*********************************************************************}
PROCEDURE Fill_File_Table;
  VAR Dirinfo : Searchrec;
      Old_Dir : String;
      Tempstr : String;
            I : BYTE;
```

```
BEGIN
  Getdir( 0, Old_Dir);
  Chdir( Path);
  Number_Files_In_Table := 0;
  Findfirst( 'H*.*', Anyfile, Dirinfo);
  WHILE ( Doserror = 0) AND (Number_Files_In_Table <= Max_Files) DO BEGIN
     Number_Files_In_Table := Number_Files_In_Table + 1;
     IF (Number_Files_In_Table > Max_Files) THEN BEGIN
        Clrscr;
        Gotoxy(1,20);
        WRITELN('Let''s not get ridiculous - over one thousand files on this');
        WRITELN('directory! Can only process the first thousand, so  put the');
        WRITELN('rest in another directory. Press any key...');
        Pause;
        Clrscr
     END
     ELSE BEGIN
        IF No_Samples_Or_Invalid_File( Dirinfo.Name) THEN
             Number_Files_In_Table:= Number_Files_In_Table - 1
        ELSE BEGIN
             File_Table [Number_Files_In_Table] :=  Dirinfo.Name;
        END;
        Findnext( Dirinfo);
     END
   END;
   IF (Number_Files_In_Table > Max_Files) THEN
      Number_Files_In_Table:= Max_Files;
   Chdir( Old_Dir);
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Show_File_Page( Num_To_Show : BYTE; Page_On : INTEGER);
   VAR Base_File_Num, File_Number : INTEGER;

BEGIN
   Clear_From( 4);
   Base_File_Num:= (Page_On - 1) * 80;
   FOR File_Number:= (Base_File_Num + 1) TO (Base_File_Num + Num_To_Show) DO
      WRITE (' [', File_Number :3, '] ', File_Table [File_Number]:11, '  ');
   IF (Num_To_Show = 80) THEN BEGIN     {More pages to come.}
       Gotoxy( 1, 25);
       WRITE('Press any key for more files...');
       Pause;
   END;
END;
{*********************************************************************}


{*********************************************************************}
PROCEDURE List_File_Table_To_Screen;

VAR  File_Page         : INTEGER;

BEGIN
  Clrscr;
  Menu_Head;
  Gotoxy (1, 3);
  WRITELN ('Meteor header files available (empty sample files ignored) :');
  FOR File_Page := 1 TO ((Number_Files_In_Table DIV 80) + 1) DO
       IF (File_Page = ((Number_Files_In_Table DIV 80) + 1)) THEN  {Last page}
            Show_File_Page( Number_Files_In_Table MOD 80, File_Page)
       ELSE Show_File_Page( 80, File_Page);
END;
{*********************************************************************}
```

```
{*****************************************************************}
FUNCTION Users_Response_To (Chars  : INTEGER; Question  : Str80) : Str8;
{-----------------------------------------------------------------}
{ This function takes as input a question stored in QUESTION, and  }
{ produces as output the user's response to the question. If CHARS }
{ is 1 then one char is read and no RETURN is necessary from the   }
{ user. If anything else, the user must press RETURN after his/her }
{ answer. The reason this was included as an INTEGER variable and  }
{ not a BOOLEAN is primarily for extension purposes. The answer is }
{ automatically mapped to upper-case in both instances.            }
{-----------------------------------------------------------------}

VAR  Answer       : Str8;
     Pos          : BYTE;
     Ans          : CHAR;

BEGIN
  Gotoxy (2,22);
  WRITE (Question, '  ');
  IF Chars = 1 THEN BEGIN
     Ans:= Upcase( Readkey);
     Answer := Ans
  END
  ELSE BEGIN                          { Read until user presses RETURN }
     Ans:= Readkey;
     IF (Ans = Esc) THEN
          Answer:= Esc
     ELSE BEGIN
          WRITE( Ans);
          READLN(Answer);
          IF (Answer[ 1] <> #14) THEN    {Backspace.}
             Answer:= Ans + Answer;
     END;
     FOR Pos:= 1 TO LENGTH( Answer) DO
         Answer[ Pos]:= Upcase( Answer[ Pos]);
  END;
  Users_Response_To := Answer;
  Gotoxy (2,22);   { Clear the question and answer from the screen }
  Clreol
END;
{*****************************************************************}



{*****************************************************************}
FUNCTION Get_User_Int (Question          : Str80;
                       Min_Int, Max_Int  : INTEGER)  : INTEGER;

VAR  Answer, Err      : INTEGER;
     Answer_Str       : Str8;

BEGIN
  REPEAT
    Answer_Str := Users_Response_To (6, Question);
    IF (Answer_Str[1] = Esc) THEN BEGIN
        Exitting:= TRUE;
        Exit;
    END;
    VAL (Answer_Str, Answer, Err);
  UNTIL (Err = 0) AND (Answer >= Min_Int) AND (Answer <= Max_Int);
  Get_User_Int := Answer
END;
{*****************************************************************}
```

```
{*****************************************************************}
FUNCTION Get_Good_Int( Lowb, Upb : INTEGER) : INTEGER;
   VAR  Ival, Err : INTEGER;
           Intstr : Str80;

BEGIN
   REPEAT
       READLN( Intstr);
       VAL( Intstr, Ival, Err);
       IF ((Err <> 0) OR (Ival < Lowb) OR (Ival > Upb)) THEN BEGIN
            Err:= 1;
            WRITE('Invalid entry - want integer in range ',Lowb,'..',Upb);
            WRITE('  - please retry : ');
       END;
   UNTIL (Err = 0);
   Get_Good_Int:= Ival
END;
{*****************************************************************}



{*****************************************************************}
PROCEDURE Get_Trail_Type_From_User (VAR Trail_Type    : BYTE);

VAR  Action                          : Str80;
     Number_Types, Total_Types       : INTEGER;
     Line_Count                      : BYTE;
     Answer                          : Str8;

BEGIN
  RESET (Abase);
  Total_Types := Filesize (Abase);
  Number_Types := 1;
  REPEAT
     Clrscr;
     Menu_Head;
     Gotoxy (1, 3);
     WRITELN ('Trail types are :');
     WRITELN;
     Line_Count := 0;
     WHILE (NOT (EOF (Abase))) AND
           (Line_Count < 15) DO
      BEGIN
        READ (Abase, Action);
        DELETE (Action, 1, 15);
        WRITELN ('  [', Number_Types :2, '] ', Action);
        Number_Types := Number_Types + 1;
        Line_Count := Line_Count + 1;
      END;
     IF (EOF (Abase)) THEN
      BEGIN
        WRITELN ('  [', Number_Types, '] ', ' UNCLASSIFIED.');
        Number_Types := 1;
        RESET (Abase);
      END;
     REPEAT
       Answer := Users_Response_To (1, 'Type M for more, C to choose type.')
     UNTIL ((Answer = 'M') OR (Answer =  'C'));
  UNTIL Answer = 'C';
  Trail_Type := Get_User_Int ('Enter type of trail to scan through :',
                              1, Total_Types + 1);
  IF Exitting THEN Exit;
  IF Trail_Type = Total_Types + 1 THEN  Trail_Type := 255
END;
{*****************************************************************}
```

```
{*********************************************************************}
FUNCTION Absolute_Val (Rval : REAL) : REAL;

BEGIN
   IF (Rval < 0.0) THEN
        Absolute_Val:= 0.0 - Rval
   ELSE Absolute_Val:= Rval
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Debug( Outl : Str80);

BEGIN
  IF Debug_Flag THEN
     WRITELN(Outl)
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Seek_Rule( Rec_Num : BYTE; VAR Rule_Back : Rules);
{Seeks and reads the rulebase to find the rule with number}
{corresponding to its REC_NUM argument.                   }

BEGIN
   SEEK( Rbase, Rec_Num);
   READ( Rbase, Rule_Back)
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Seek_Cond( Rec_Num : BYTE; VAR Cond_Back : Str80);
{Seeks and reads a string from the condition base.}

BEGIN
   SEEK(Cbase, Rec_Num);
   READ(Cbase,Cond_Back)
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Seek_Act( Rec_Num : BYTE; VAR Act_Back : Str80);
{Seeks and reads a string from the action base.}

BEGIN
   SEEK(Abase, Rec_Num);
   READ(Abase,Act_Back)
END;
{*********************************************************************}


{*********************************************************************}
FUNCTION Conds_In (Some_Rule : Rules) : BYTE;
{Returns the number of conditions currently in the rule passed to it.}

BEGIN
   Conds_In:= Some_Rule[0].Cond_Num
END;
{*********************************************************************}
```

```
{*********************************************************************}
FUNCTION Eo_St(St_In : Str80; Pos : BYTE) : BOOLEAN;
{Checks if at end of string - if POS > length of string.}

BEGIN
   IF (Pos <= LENGTH(St_In)) THEN
         Eo_St:= FALSE
   ELSE Eo_St:= TRUE
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Get_Item( VAR Item : Str80; VAR Pos : BYTE; St_In : Str80);
{Returns a string containing the next item from POS in ST_IN. An item}
{is defined here as either being a  group  of  characters  in  SPACES}
{- this contains characters ' ', '.', ',', and ';' - or  a  group  of}
{characters containg no character in SPACES.                         }

BEGIN
  Item:= '';
  IF (St_In[Pos] IN Spaces) THEN
      WHILE ((St_In[Pos] IN Spaces) AND (NOT Eo_St(St_In,Pos))) DO BEGIN
         Item:= Item + St_In[Pos];
         Pos:= Pos + 1
      END
  ELSE
     WHILE ((NOT(St_In[Pos] IN Spaces)) AND (NOT Eo_St(St_In,Pos))) DO BEGIN
           Item:= Item + St_In[Pos];
           Pos:= Pos + 1
     END
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Pause_It;
{Delay to allow user to read current message.}
  VAR Anykey : CHAR;

BEGIN
   Gotoxy(1,25);
   Clreol;
   WRITE('Press any key to continue.');
   Anykey:= Readkey;
   Gotoxy(1,25);
   Clreol
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Wait_Till_Read;
{Delay used for the various help menus.}

BEGIN
    Gotoxy(1,25);
    Normvideo;
    WRITE('Press any key when you've finished reading this.');
    Lowvideo;
    Anykey:= Readkey;
END;
{*********************************************************************}
```

```
{*********************************************************************}
PROCEDURE Alarm( Move : BOOLEAN);
{Sends 'bells' on fatal error to alert user.}
  VAR I, J : INTEGER;

BEGIN
   IF Move THEN Gotoxy(1,25);
   FOR I:= 1 TO 5 DO BEGIN
       WRITE(CHR(7));
       FOR J:= 1 TO 4000 DO {Delay};
   END
END;
{*********************************************************************}



{*********************************************************************}
FUNCTION Get_Valid_Choice_From( Legalset : Charset) : CHAR;
{For use with the various menus and options, this reads characters}
{from keyboard, (so no screen echo), until a  character  is  found}
{which is in the LEGALSET parameter. The uppercase version of this}
{character is the function result. (Allows user to work  in either}
{case, while program just need worry about upper case after this.)}
   VAR Local_Char : CHAR;

BEGIN
   REPEAT
      Local_Char:= Readkey;
      Local_Char:= Upcase(Local_Char)
   UNTIL (Local_Char IN Legalset);
   Get_Valid_Choice_From:= Local_Char
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Help_Head;
{Used to initialise 'help' explanation screens.}

BEGIN
  Clrscr;
  Normvideo;
  WRITELN('HELP  - for more detail see "The User's Guide to TrailStar."');
  WRITELN
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Menu_Write( St_In : Str80);
{Writes a string in 'menu' fashion. Does this by highlighting first}
{letter of string, (the option character). eg. If passed the string}
{"A =  Abort program", it would print out a  highlighted "A",  then}
{the rest of the string at lower intensity.                         }

BEGIN
   Normvideo;
   WRITE(St_In[1]);
   St_In[1]:= ' ';
   Lowvideo;
   WRITELN(St_In)
END;
{*********************************************************************}
```

```
{*********************************************************************}
PROCEDURE Print_Ordinal( Parm_Num : BYTE);
{Used to print parameters specifying ordinal positions.}
   VAR Position : Str12;

BEGIN
   Parm_Num:= Parm_Num - 100;      {From 101 -> 200  to  1 -> 100.}
   STR(Parm_Num, Position);
   IF (Parm_Num > 19) THEN
       Parm_Num:= (Parm_Num MOD 20);
   CASE Parm_Num OF
          0 : Position:= Position + 'th';
          1 : Position:= Position + 'st';
          2 : Position:= Position + 'nd ';
          3 : Position:= Position + 'rd ';
     4..19 : Position:= Position + 'th '
   END;
   WRITE(Position)
END;
{*********************************************************************}


{*********************************************************************}
PROCEDURE Print_Parm( Parm_Num : BYTE);
{Prints out an intelligible (to user) version of a particular parameter.}

BEGIN
   IF (Parm_Num < 101) THEN                {Cardinal value.}
        WRITE( Parm_Num)
   ELSE
        IF (Parm_Num < 201) THEN           {Ordinal value. }
            Print_Ordinal( Parm_Num)
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Show_Cond( Cond_In : Condition);
{Writes a condition to screen, substituting text describing}
{the actual parameter used for  any  parameter  flags, ("!"}
{symbols), found in the condition base text.                }
  VAR Parm_Count, Pos : BYTE;
        Constr, Item : Str80;
             Cnd_Num : INTEGER;

BEGIN
  Cnd_Num:= Cond_In.Cond_Num;
  Seek_Cond((Cnd_Num - 1),Constr);
  Pos:= 1;
  Parm_Count:= 0;
  WHILE (NOT (Eo_St(Constr,Pos))) DO BEGIN
        Get_Item(Item,Pos,Constr);
        IF (Item[1] = '!') THEN BEGIN
             Parm_Count:= Parm_Count + 1;
             Print_Parm( Cond_In.C_Parms[ Parm_Count])
        END
        ELSE WRITE(Item)
  END;
  WRITELN
END;
{*********************************************************************}
```

```
{*********************************************************************}
PROCEDURE Write_Conds ( Rule_Containing : Rules);
{Writes all the conditions in a particular rule to screen.}
   VAR Cond_Count, I : BYTE;

BEGIN
   Cond_Count:= Rule_Containing[0].Cond_Num;
   FOR I:= 1 TO Cond_Count DO
       Show_Cond( Rule_Containing[I])
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Show_Whynot_Act( Act_In : Condition);
{Writes action text to screen, substituting text describing the}
{actual parameter used in the action for parameter flags in the}
{action base text.                                                  }
  VAR Parm_Count, Pos : BYTE;
         Actstr, Item : Str80;
              Act_Num : INTEGER;

BEGIN
  Act_Num:= Act_In.Cond_Num;
  Seek_Act((Act_Num - 1),Actstr);
  Pos:= 1;
  Parm_Count:= 0;
  WHILE (NOT (Eo_St(Actstr,Pos))) DO BEGIN
        Get_Item(Item,Pos,Actstr);
        IF (Item = 'Trail') OR (Item = 'typed') OR (Item = 'as') THEN
            {Do nothing}
        ELSE BEGIN
             IF (Item[1] = '!') THEN BEGIN
                Parm_Count:= Parm_Count + 1;
                Print_Parm( Act_In.C_Parms[ Parm_Count])
             END
             ELSE WRITE(Item)
        END;
  END;
  WRITELN
END;
{*********************************************************************}


{*********************************************************************}
PROCEDURE Show_Act( Act_In : Condition);
{Writes action text to screen, substituting text describing the}
{actual parameter used in the action for parameter flags in the}
{action base text.                                                  }
  VAR Parm_Count, Pos : BYTE;
         Actstr, Item : Str80;
              Act_Num : INTEGER;

BEGIN
  Act_Num:= Act_In.Cond_Num;
  Seek_Act((Act_Num - 1),Actstr);
  Pos:= 1;
  Parm_Count:= 0;
  WHILE (NOT (Eo_St(Actstr,Pos))) DO BEGIN
        Get_Item(Item,Pos,Actstr);
        IF (Item[1] = '!') THEN BEGIN
             Parm_Count:= Parm_Count + 1;
             Print_Parm( Act_In.C_Parms[ Parm_Count])
        END
        ELSE WRITE(Item)
  END;
  WRITELN
END;
{*********************************************************************}
```

```
{*********************************************************************}
FUNCTION Float (INT   : INTEGER)   : REAL;

BEGIN
  Float := INT
END;
{*********************************************************************}



{*********************************************************************}
FUNCTION Round_Up (Reel   : REAL)   : INTEGER;

BEGIN
  IF Reel > TRUNC (Reel) THEN  Round_Up := TRUNC (Reel) + 1
  ELSE                         Round_Up := TRUNC (Reel)
END;
{*********************************************************************}



{*********************************************************************}
FUNCTION Number_Digits (Value   : INTEGER)   : BYTE;

{-------------------------------------------------------}
{ This Function will return the number of digits in VALUE }
{-------------------------------------------------------}

VAR  Digits, Extras          : INTEGER;
     Accumulated_Value       : REAL;

BEGIN
  IF Value < 0 THEN  Extras := 1
  ELSE               Extras := 0;
  Value := Abs (Value);
  Accumulated_Value := 9.999999999999;
  Digits := 1;
  WHILE Accumulated_Value < Value DO
   BEGIN
     Digits := SUCC (Digits);
     Accumulated_Value := Accumulated_Value * 10.0
   END;
  Number_Digits := Digits + Extras
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Kill_Lead_Zeroes( VAR Str_In : Str12);
{Deletes lead zeroes from string representation of a number.}
   VAR Temp_Str : Str12;
         Pos, I : INTEGER;
BEGIN
    Pos:= 1;
    Temp_Str:= '';
    WHILE ((Pos < LENGTH(Str_In)) AND (Str_In[ Pos] = '0')) DO
          Pos:= Pos + 1;
    FOR I:= Pos TO LENGTH(Str_In) DO
        Temp_Str:= Temp_Str + Str_In[I];
    Str_In:= Temp_Str
END;
{*********************************************************************}
```

```
{*****************************************************************}
PROCEDURE Get_Hour_File_Descrip( Name_Of_Hour_File : Str12; VAR Systm,
                                 Day_Of, Hour_Of, Year_Of : Str12);
{Gets substrings from an hour file name to return system, day, hour.}
      VAR Pos, Ofset : INTEGER;

BEGIN
   IF (Name_Of_Hour_File[ 2] = ':') THEN
        Ofset:= 2
   ELSE Ofset:= 0;
   Systm:= Name_Of_Hour_File[3 + Ofset];
   Day_Of:= Name_Of_Hour_File;
   IF (LENGTH( Day_Of) < 10) THEN BEGIN
       DELETE( Day_Of, 1, (3 + Ofset));
       DELETE( Day_Of, 4, 7);
       Year_Of:= 'Pre-89';
   END
   ELSE BEGIN
       Year_Of:= Day_Of;
       DELETE( Year_Of, 1, (3 + Ofset));
       DELETE( Year_Of, 3, 9);
       IF (Year_Of[1] IN ['8','9']) THEN
            Year_Of:= '19' + Year_Of
       ELSE Year_Of:= '20' + Year_Of;
       DELETE( Day_Of, 1, (5 + Ofset));
       DELETE( Day_Of, 4,7);
   END;
   Kill_Lead_Zeroes( Day_Of);
   Hour_Of:= Name_Of_Hour_File;
   Ofset:= 6;
   WHILE (Hour_Of[ Ofset] <> '.') DO
        Ofset:= Ofset + 1;
   Hour_Of:= '';
   FOR Pos:= (Ofset + 1) TO LENGTH( Name_Of_Hour_File) DO
        Hour_Of:= Hour_Of + Name_Of_Hour_File[ Pos];
   Kill_Lead_Zeroes( Hour_Of)
END;
{*****************************************************************}



{*****************************************************************}
PROCEDURE Fit_Lines_To_Trail (Start, Stop          : INTEGER;
                              Variance_Tolerance   : REAL;
                              Last_Line            : INTEGER);


VAR  Last_Pos, First_Pos, Sample, Sample_Increment   : INTEGER;
     Line_Number                                     : INTEGER;
     Last_B0, Last_B1, B0, B1, Variance, X_Scale     : REAL;
     New_Div, New_Y_Bar                              : REAL;

      i   : INTEGER;
      junk  : CHAR;
```

```
{*****************************************************************}
PROCEDURE Fit_Line (    First_Pos, Last_Pos    : INTEGER;
                    VAR B0, B1                 : REAL);

VAR  X_Bar, Dividend        : REAL;
     Last_First_Gap         : INTEGER;
       Samp, P1, P2         : INTEGER;

BEGIN
  P1:= First_Pos;  P2:= Last_Pos;  Samp:= Sample;
  Last_First_Gap := Last_Pos - First_Pos;
  X_Bar := First_Pos + (Last_First_Gap) / 2;
  New_Y_Bar := New_Y_Bar + (Trail_Array [Last_Pos] - New_Y_Bar) /
      Float (Last_First_Gap + 1);

  Dividend := 0.0;
  New_Div := New_Div + (Sqr (Float (Last_First_Gap))  +
             (Last_First_Gap)) / 4.0;
  FOR Samp:= P1 TO P2 DO
   BEGIN
     Dividend :=  Dividend + (Samp - X_Bar) *
        (Trail_Array [Samp] - New_Y_Bar);
   END;
  B1 := Dividend / New_Div;
  B0 := New_Y_Bar - (B1 * X_Bar)
END;
{*****************************************************************}


{*****************************************************************}
FUNCTION Get_Variance (First_Pos, Last_Pos  : INTEGER;
                       B0, B1               : REAL)  : REAL;

VAR  Sum_Of_Squares, Expected_Value     : REAL;
     Sample                             : INTEGER;

BEGIN
  Sum_Of_Squares := 0.0;
  FOR Sample := First_Pos TO Last_Pos DO
   BEGIN
     Expected_Value := B0 + B1 * Sample;
     Sum_Of_Squares := Sum_Of_Squares +
        Sqr (Trail_Array [Sample] - Expected_Value)
   END;
  Get_Variance := Sum_Of_Squares / (Last_Pos - First_Pos)
END;
{*****************************************************************}
```

```
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
BEGIN
  Line_Number := 0;
  Fillchar (Lines, Sizeof (Lines), 0);
  Last_Pos := Start;
  Last_B0 := Float (Trail_Array [Start]);
  Last_B1 := 0.0;
  WHILE (Last_Pos <= Stop) AND (Line_Number <= Last_Line) DO
   BEGIN
     First_Pos := Last_Pos;
     New_Y_Bar := Trail_Array [First_Pos];
     New_Div := 0.0;
     REPEAT
       Last_Pos := Last_Pos + 1;
       Last_B0 := B0;
       Last_B1 := B1;
       Fit_Line (First_Pos, Last_Pos, B0, B1);
       Variance := Get_Variance (First_Pos, Last_Pos, B0, B1)
     UNTIL (Variance > Variance_Tolerance) OR (Last_Pos > Stop);
     Line_Number := Line_Number + 1;
     IF Debug_Flag THEN
      BEGIN
        Gotoxy (1,1);
        WRITE ('Fitted line :  ', Line_Number)
      END;
     Lines [Line_Number].Start := First_Pos;
     Lines [Line_Number].Stop  := Last_Pos - 1;
     Lines [Line_Number].B0    := Last_B0;
     Lines [Line_Number].B1    := Last_B1;
   END;
  Number_Of_Lines := Line_Number;
  IF Debug_Flag THEN
   BEGIN
     WRITELN;
     FOR I := 1 TO Number_Of_Lines DO
       WRITELN ('B1 =  ', Lines [i].B1);
     Junk:= Readkey;
     Clrscr
   END
END;
{*********************************************************************}




{*********************************************************************}
PROCEDURE Stu_Parabola( Startp, Endp : INTEGER);
 VAR Amat : ARRAY[1..3,1..3] OF REAL;
     Cvector : ARRAY[1..3] OF REAL;



{*********************************************************************}
FUNCTION Powerit( Value, Nthpower : INTEGER) : REAL;
   VAR Run_Tot:REAL;
           K   : INTEGER;

BEGIN
   Nthpower:= Nthpower - 1;
   IF (Nthpower = 0) THEN
       Powerit:= 1
   ELSE BEGIN
       Run_Tot:= 1;
       FOR K:= 1 TO Nthpower DO
           Run_Tot:= Run_Tot * Value;
       Powerit:= Run_Tot
   END
END;
{*********************************************************************}
```

```
{*****************************************************************}
PROCEDURE Work_Out_Array_Entries;
    VAR I, J, Svalue : INTEGER;

BEGIN
   FOR I:= 1 TO 3 DO BEGIN
     FOR J:= 1 TO 3 DO
        Amat[I,J]:= 0;
     Cvector[I]:= 0;
   END;
   FOR I:= 1 TO 3 DO
     FOR J:= 1 TO 3 DO
        FOR Svalue:= Startp TO Endp DO
         Amat[I,J]:= Amat[I,J] + Powerit(Svalue , I) * Powerit(Svalue ,J);
   FOR I:= 1 TO 3 DO
     FOR Svalue:= Startp TO Endp DO
       Cvector[I]:=
         Cvector[I] + (Trail_Array[Svalue] - 140) * Powerit(Svalue ,I);
END;
{*****************************************************************}


{*****************************************************************}
PROCEDURE Solve_By_Gauss;
   VAR Gmat : ARRAY[1..3,1..4] OF REAL;
       I, J, Cur_Col, Piv_Row, Row_Under : INTEGER;
         Divisor, Multip : REAL;

BEGIN
  FOR I:= 1 TO 3 DO BEGIN
     FOR J:= 1 TO 3 DO
        Gmat[I,J]:= Amat[I,J];
     Gmat[I,4]:= Cvector[I]
  END;
  {!!!}
  IF Debug_Flag THEN BEGIN
     Clrscr;
     WRITELN('ORIGINAL GMATRIX');
     FOR I:= 1 TO 3 DO BEGIN
         FOR J:= 1 TO 4 DO
             WRITE(Gmat[I,J]:8:4,'    ');
         WRITELN
    END
  END;
  IF Debug_Flag THEN  WRITELN;
  FOR Piv_Row:= 1 TO 3 DO BEGIN
      Divisor:= Gmat[ Piv_Row, Piv_Row];
      IF (Divisor <> 0.0) THEN BEGIN
        FOR Cur_Col:= 1 TO 4 DO
           Gmat[ Piv_Row, Cur_Col]:= Gmat[ Piv_Row, Cur_Col] / Divisor;
        {Getting first entry to 1.}
        FOR Row_Under:= (Piv_Row + 1) TO 3 DO BEGIN
            Multip:= Gmat[ Row_Under, Piv_Row];
            FOR Cur_Col:= Piv_Row TO 4 DO
                Gmat[ Row_Under, Cur_Col]:=
                  Gmat[ Row_Under, Cur_Col] - (Multip * Gmat[Piv_Row, Cur_Col]);
       END;
      END;
  END;
  IF Debug_Flag THEN
   BEGIN
     WRITELN('REDUCED GMATRIX');
     FOR I:= 1 TO 3 DO BEGIN
         FOR J:= 1 TO 4 DO
             WRITE(Gmat[I,J]:8:4,'    ');
         WRITELN
    END
  END;
  IF debug_flag THEN  WRITELN;
  A_Coeff:= Gmat[ 3,4];      {As mat[3,3] = 1, mat[3,2] and [3,1] = 0.}
  B_Coeff:= Gmat[ 2,4] - Gmat[2,3] * A_Coeff;
  C_Coeff:= Gmat[ 1,4] - Gmat[1,2] * B_Coeff - Gmat[1,3] * A_Coeff;
END;
{*****************************************************************}
```

```
{*****************************************************************}
PROCEDURE Get_Parb_Variance;
  VAR Expected_Val, Sqr_Dif : REAL;
                    Curpos : INTEGER;

BEGIN
  Sqr_Dif:= 0.0;
  FOR Curpos:= Startp TO Endp DO BEGIN
      Expected_Val:= A_Coeff * (Curpos ) * (Curpos ) +
                     B_Coeff * (Curpos ) + C_Coeff;
      Sqr_Dif:= Sqr_Dif + Sqr( (Trail_Array[Curpos] - 140.0) - Expected_Val);
  END;
  Parab_Var:= Sqr_Dif/(Endp - Startp + 1)
END;
{*****************************************************************}



{*****************************************************************}
PROCEDURE Write_Results;

BEGIN
   WRITE('Parabola is ',A_Coeff,'X*X + ',B_Coeff,'X + ',C_Coeff,'.');
   WRITELN;
   WRITE('Variance of parabola is ');
   WRITELN(Parab_Var:10:3);
   WRITELN(B_Coeff * Trail_Array[0]);
   Anykey:= Readkey;
   Clrscr
END;
{*****************************************************************}



BEGIN
  Work_Out_Array_Entries;
  Solve_By_Gauss;
  Get_Parb_Variance;
  IF Debug_Flag THEN  Write_Results
END;
{*****************************************************************}



{*****************************************************************}
FUNCTION One_Time( Inrec : Time_Rec) : REAL;

BEGIN
  WITH Inrec DO BEGIN
     One_Time:=
        Huns + (100.0 * (Secs + (60.0 * (Mins + (60.0 * Hours)))));
  END
END;
{*****************************************************************}


{*****************************************************************}
FUNCTION Variance_From_Line( B_Zero, B_One : REAL;
                             Posa, Posn : INTEGER) : REAL;
    VAR Expected_Value, Square_Sum : REAL;
                                 I : INTEGER;

BEGIN
   Square_Sum:= 0.0;
   FOR I;= Posa TO Posn DO BEGIN
       Expected_Value:= B_Zero + (B_One * I);
       Square_Sum:= Square_Sum + Sqr((Trail_Array[I] - 140) - Expected_Value)
   END;
   IF (Posn <> Posa) THEN
        Variance_From_Line:= Square_Sum/(Posn - Posa)
   ELSE Variance_From_Line:= 0.0
END;
{*****************************************************************}
```

```
{*********************************************************************}
PROCEDURE Linear_Regression_On_Part_Of_Trail( Startp, Endp : INTEGER);
   VAR Top_Sum, Low_Sum : REAL;
                      I : INTEGER;

BEGIN
     Part_X_Bar:= (Startp + Endp)/2;
     Part_Y_Bar:= 0.0;
     FOR I:= Startp TO Endp DO
         Part_Y_Bar:= Part_Y_Bar + (Trail_Array[I] - 140.0);
     Part_Y_Bar:= Part_Y_Bar/(Endp + 1 - Startp);
     Low_Sum:= 0.0;
     Top_Sum:= 0.0;
     FOR I:= Startp TO Endp DO BEGIN
         Top_Sum:= Top_Sum +
              ((I - Part_X_Bar) * ((Trail_Array[I] - 140) - Part_Y_Bar));
         Low_Sum:= Low_Sum + ((I - Part_X_Bar) * (I - Part_X_Bar));
     END;
     IF (Low_Sum = 0.0) THEN
          Part_B1:= 0.0
     ELSE Part_B1:= Top_Sum/Low_Sum;
     Part_B0:= Part_Y_Bar - (Part_B1 * Part_X_Bar);
     Part_Variance:= Variance_From_Line( Part_B0, Part_B1, Startp, Endp);
END;
{*********************************************************************}



{*********************************************************************}
PROCEDURE Change_Base;
  VAR I : INTEGER;

BEGIN
  Menu_Head;
  WRITELN('           RULE BASE SELECTION');
  IF (Basenum <> 0) THEN BEGIN
     CLOSE( Pribase);
     CLOSE( Rbase);
  END;
  WRITELN;
  WRITELN('Available rule bases are as follows : ');
  FOR I:= 1 TO Topbase DO
      IF (I IN Bases_Available) THEN
          WRITE( I:2, '  ');
  WRITELN;
  REPEAT
      Clear_From( 22);
      WRITE('Please enter new base : ');
      Basenum:= Get_Good_Int( 1, Topbase);
      IF (NOT (Basenum IN Bases_Available)) THEN BEGIN
          WRITELN;
          WRITE('No such base available! Press any key to try again...');
          Pause;
          Basenum:= 0;
      END;
   UNTIL (Basenum <> 0);
   STR( Basenum, Curbase);
   ASSIGN( Rbase, 'RULES' + Curbase + '.STA');
   RESET( Rbase);
   ASSIGN( Pribase, 'PRIOR' + Curbase + '.STA');
   RESET( Pribase);
   Clrscr;
END;
{*********************************************************************}


END.
```

```
UNIT Ts2;
{This unit contains the action determination (classification}
{assignment) routine.                                        }

Interface

USES Ts1;

PROCEDURE Determine_Action( Action_Part : Condition;
                            VAR Type_Back : BYTE);

IMPLEMENTATION

{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Determine_Action( Action_Part : Condition;
                            VAR Type_Back : BYTE);


   (* Note : Action_Part OF Form :
         RECORD
             Cond_Num : BYTE;
              C_Parms : ARRAY[1..3] OF BYTE
         END                                   *)

{  Act 1 - Bad data
       2 - Short, mid-peak
       3 - Short mush
       4 - Medium time mid-peak
       5 - Low, early peak classic
       6 - Flat bell
       7 - Straight line mush - medium
       8 - Straight line mush - long
       9 - Classic underdense
      10 - Classic with plateau
      11 - Classic round-top
      12 - Classic late fall (bad initial fall)
      13 - Classic notched rise
      14 - Classic drunk rise
      15 - True bell
      16 - Multi-plateau classic
      17 - Multi-slope classic
      18 - Twins
      19 - Square root sign
      20 - Overdense rectified sine wave
      21 - Overdense non-sine
      22 - Gothic rocker
      23 - Downward tending mush
      24 - Hazy classic
      25 - Extension mush
      26 - Extension overdense
      27 - Sinusoidal overdense
      28 - Wind-blown overdense
      29 - Hump-backed classic     }


     VAR Place_To_Write : INTEGER;
                Act_Num : BYTE;


BEGIN
   Act_Num:= Action_Part.Cond_Num;
   Type_Back:= Act_Num;
   Hrec.Trail_Type:= Type_Back;
   Hrec.Useful_Time[ Plus6, 1]:= Basenum;
   IF (EOF (Trail_Head)) THEN
       Place_To_Write:= Filesize( Trail_Head) - 1
   ELSE Place_To_Write:= Filepos( Trail_Head) - 1;
   SEEK( Trail_Head, Place_To_Write);
   WRITE( Trail_Head, Hrec)
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}

END.
```

```
UNIT Ts3;
{This unit contains the condition checking routines.}

Interface

USES Ts1, Ts2;


FUNCTION Condition_Checker( Cond_In : Condition) : BOOLEAN;

IMPLEMENTATION


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Condition_Checker( Cond_In : Condition) : BOOLEAN;
   VAR Cnd_Num : INTEGER;
   {!!!!! ATTENTION : HIGH-LEVEL MODIFIER. !!!!!}
   {! This routine must be updated to have calls}
   {! to any new condition routines you enter.  }

   (* Note : Cond_In OF Form :
           RECORD
                Cond_Num : BYTE;
                C_Parms : ARRAY[1..3] OF BYTE
            END                                    *)


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond1 : BOOLEAN;
{Dummy condition for default rules.}

BEGIN
   Cond1:= TRUE
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond2 : BOOLEAN;
{Checks if straight line variance over entire trail is greater}
{than <Number> times variance on straight line through fall.  }
      VAR N_Times_Fall : REAL;

BEGIN
   N_Times_Fall:= Down_Variance * Cond_In.C_Parms[1];
   Cond2:= (Variance > N_Times_Fall);
   IF User_Wants_Actual_Condition_Value THEN BEGIN
      IF (Down_Variance <> 0) THEN
           WRITE('Trail variance is ',(Variance/Down_Variance):1:2,' times')
      ELSE WRITE('Trail variance is infinitely');
      WRITELN(' greater than fall variance.');
      WRITE('(Trail variance : ',Variance:1:2);
      WRITELN('  Fall variance : ',Down_Variance:1:2,')');
   END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond3 : BOOLEAN;
{Checks if straight line variance over  entire trail is  less or}
{equal to <Number> times variance on straight line through fall.}

BEGIN
   Cond3:= (NOT Cond2);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond4 : BOOLEAN;
{Checks if straight line variance over entire trail is greater}
{than <Number> times variance on straight line through rise.  }
      VAR N_Times_Rise : REAL;

BEGIN
   N_Times_Rise:= Up_Variance * Cond_In.C_Parms[1];
   Cond4:= (Variance > N_Times_Rise);
   IF User_Wants_Actual_Condition_Value THEN BEGIN
      IF (Up_Variance <> 0) THEN
           WRITE('Trail variance is ',(Variance/Up_Variance):1:2,' times')
      ELSE WRITE('Trail variance is infinitely');
      WRITELN(' greater than rise variance.');
      WRITE('(Trail variance : ',Variance:1:2);
      WRITELN('  Rise variance : ',Up_Variance:1:2,')');
   END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond5 : BOOLEAN;
{Checks if straight line variance over  entire trail is  less or}
{equal to <Number> times variance on straight line through rise.}

BEGIN
   Cond5:= NOT( Cond4);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond6 : BOOLEAN;
{Checks if duration greater than <number> times 50 mSec.}
   VAR Duration_On_Sample : REAL;

BEGIN
    Duration_On_Sample:= Trail_Array[0] * 5;  {Num samples * 5}
    IF (Duration_On_Sample > (Cond_In.C_Parms[1] * 50.0)) THEN
         Cond6:= TRUE
    ELSE Cond6:= FALSE;
    IF User_Wants_Actual_Condition_Value THEN
         WRITELN('Duration was ',Duration_On_Sample:1:2,' milliseconds.');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond7 : BOOLEAN;
{Checks if duration less or equal to <number> times 50 mSec.}

BEGIN
   Cond7:= NOT (Cond6)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond8 : BOOLEAN;
{Checks if upper plateau present.}

BEGIN
   Cond8:= (Plat_Count <> 0)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond9 : BOOLEAN;
{Checks if no upper plateau present.}

BEGIN
  Cond9:= (Plat_Count = 0)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond10 : BOOLEAN;
{Checks if peak in <Xth> third of trail.}
   VAR Peak_Place, One_Third : REAL;
                  Check_Third : BYTE;

BEGIN
   Check_Third:= Cond_In.C_Parms[1] - 100;
   Peak_Place:= Peak_Pos * 5.0;
   One_Third:= (Trail_Array[0] * 5.0)/3.0;
   IF (Peak_Place < One_Third) THEN
        Cond10:= (Check_Third = 1)
   ELSE
       IF (Peak_Place < (One_Third + One_Third)) THEN
            Cond10:= (Check_Third = 2)
       ELSE Cond10:= (Check_Third = 3);
   IF User_Wants_Actual_Condition_Value THEN BEGIN
       IF (Peak_Place < One_Third) THEN
           WRITELN('Peak was in first third of trail.')
       ELSE
          IF (Peak_Place < (One_Third + One_Third)) THEN
              WRITELN('Peak was in second third of trail.')
          ELSE WRITELN('Peak was in last third of the trail.');
   END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond11 : BOOLEAN;
{Checks if amplitude range over trail is greater than <number> dBm.}

BEGIN
   IF ((High_Tide - Low_Tide) > Cond_In.C_Parms[1]) THEN
        Cond11:= TRUE
   ELSE Cond11:= FALSE;
   IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Trail amplitude range was ',(High_Tide - Low_Tide),' dBm');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond12 : BOOLEAN;
{Checks if amplitude range over trail is less or equal to <number> dBm.}

BEGIN
   Cond12:= NOT( Cond11)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond13 : BOOLEAN;
{Checks if peak greater than (<number> - 130) dBm.}

BEGIN
   IF (High_Tide > (Cond_In.C_Parms[1] - 130)) THEN
        Cond13:= TRUE
   ELSE Cond13:= FALSE;
   IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Peak amplitude was ',High_Tide,' dBm');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond14 : BOOLEAN;
{Checks if peak less or equal to (<number> - 130) dBm.}

BEGIN
   Cond14:= NOT( Cond13)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond15 : BOOLEAN;
{Checks if time constant on fall greater than <number> * 0.01.}

BEGIN
   IF (Tau > (Cond_In.C_Parms[1] * 0.01)) THEN
        Cond15:= TRUE
   ELSE Cond15:= FALSE;
    IF User_Wants_Actual_Condition_Value THEN
        WRITELN('Time constant (Tau) on fall was ',Tau:1:2);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond16 : BOOLEAN;
{Checks if time constant on fall less or equal to <number> * 0.01.}

BEGIN
   Cond16:= NOT( Cond15)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond17 : BOOLEAN;
{Checks if trail variance from straight line greater than <number> * 0.1.}

BEGIN
   Cond17:= (Variance > (Cond_In.C_Parms[1] * 0.1));
    IF User_Wants_Actual_Condition_Value THEN BEGIN
        WRITE('Variance from straight line fit over whole trail was ');
        WRITELN( Variance:1:2);
    END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond18 : BOOLEAN;
{Checks if variance <= <number * 0.1.}

BEGIN
   Cond18:= NOT( Cond17)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond19 : BOOLEAN;
{Checks if upper plateau comprises > <number>% of trail.}
    VAR Tsize, Psize : REAL;

BEGIN
   IF (Plat_Count = 0) THEN
       Cond19:= FALSE
   ELSE BEGIN
        Tsize:= Trail_Array[0] * 5.0;
        Psize:= (Plateaus[ Highest_Platt, Eplatt] -
                   Plateaus[ Highest_Platt, Splatt] + 1) * 5.0;
        Cond19:= ((Psize/Tsize * 100.0) > Cond_In.C_Parms[1])
   END;
   IF User_Wants_Actual_Condition_Value THEN BEGIN
       IF (Plat_Count = 0) THEN
            WRITELN('No upper plateau in trail.')
       ELSE BEGIN
            WRITELN('Plateau covered ',((Psize/Tsize) * 100.0):1:2);
            WRITELN('% of the trail.');
       END;
    END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond20 : BOOLEAN;

BEGIN
  Cond20:= NOT( Cond19)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond21 : BOOLEAN;
{Number of upper plateaus > <number>}

BEGIN
  Cond21:= (Plat_Count > Cond_In.C_Parms[1]);
  IF User_Wants_Actual_Condition_Value THEN
      WRITELN('There were ',Plat_Count,' upper plateaus.');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond22 : BOOLEAN;
{Number of upper plateaus <= <number>}

BEGIN
  Cond22:= (Plat_Count <= Cond_In.C_Parms[1])
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond23 : BOOLEAN;
{Greater than <number> fades, each greater than <number> * 10mSec.}
   VAR Num_Fades, Num_Long_Fades, Ex_Count : INTEGER;
                                Fade_Size : REAL;

BEGIN
  Num_Fades:= Num_Minima - 1;
  IF (Num_Fades <= Cond_In.C_Parms[1]) THEN
       Cond23:= FALSE
  ELSE BEGIN
       Num_Long_Fades:= 0;
       Ex_Count:= 1;
       WHILE ((Ex_Count+1) <= Num_Minima) DO BEGIN
              Fade_Size:=
                 Minima[ Ex_Count + 1, Xval] - Minima[ Ex_Count, Xval] + 1;
               IF (Fade_Size > (Cond_In.C_Parms[2] * 2.0)) THEN
                   Num_Long_Fades:= Num_Long_Fades + 1;
               Ex_Count:= Ex_Count + 1
       END;
       Cond23:= (Num_Long_Fades > Cond_In.C_Parms[1])
 END;
 IF User_Wants_Actual_Condition_Value THEN BEGIN
     IF (Num_Fades < Cond_In.C_Parms[1]) THEN
          WRITELN('There were only ',Cond_In.C_Parms[1],' fades in total.')
     ELSE BEGIN
          WRITE('There were ',Num_Long_Fades,' fades lasting longer ');
          WRITELN('then ',(Cond_In.C_Parms[ 2] * 10.0):1:0,' milliseconds');
     END
 END
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond24 : BOOLEAN;
{Less or equal to <number> fades, each greater than <number> * 10mSec.}

BEGIN
  Cond24:= NOT( Cond23)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond25 : BOOLEAN;
{Data is unreasonable.}

BEGIN
  Cond25:= Garbage_Flag
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond26 : BOOLEAN;
{Peak is in <Xth> <number> % of trail.}
  VAR Peak_Place, Region_Size : REAL;
           Start_Reg, End_Reg : REAL;
                 Check_Region : INTEGER;

BEGIN
   Check_Region:= Cond_In.C_Parms[1] - 100;    {1st, 2nd, etc.}
   Region_Size:= (Trail_Array[ 0] * 5.0) * Cond_In.C_Parms[2] / 100.0;
   {Region = number % of trail size.}
   Peak_Place:= Peak_Pos * 5.0;
   Start_Reg:= (Check_Region - 1) * Region_Size;
   End_Reg:= Check_Region * Region_Size;
   IF ((Start_Reg <= Peak_Place) AND (Peak_Place < End_Reg)) THEN
        Cond26:= TRUE
   ELSE Cond26:= FALSE;
   IF User_Wants_Actual_Condition_Value THEN BEGIN
       WRITE('Peak is found after ',((Peak_Pos/Trail_Array[0]) * 100.0):1:0);
       WRITELN('% of the trail.');
   END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond27 : BOOLEAN;
{Peak not in <Xth> <number> % of trail.}

BEGIN
  Cond27:= NOT( Cond26)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond28 : BOOLEAN;
{Fall good in last <number> % of fall.}
  VAR Fall_Length, First_Bit : REAL;
    Startpos, Int_First_Bit : INTEGER;

BEGIN
  {First find what area of fall dealing with.}
  Fall_Length:= (Trail_Array[0] - Fallpos) * 5.0;
  First_Bit:= (100.0 - Cond_In.C_Parms[1]);     {ie. % not considered.}
  First_Bit:= Fall_Length * First_Bit / 100.0;
  Int_First_Bit:= TRUNC( First_Bit / 5.0);
  Startpos:= Peak_Pos + Int_First_Bit;
  Linear_Regression_On_Part_Of_Trail(  Startpos, Trail_Array[0]);
  IF (Variance > (3.0 * Part_Variance)) THEN
       Cond28:= TRUE
  ELSE Cond28:= FALSE;
  IF User_Wants_Actual_Condition_Value THEN BEGIN
      WRITELN('Variance in this section was ',Part_Variance:1:2);
      WRITELN('while variance over trail was ',Variance:1:2);
  END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond29 : BOOLEAN;
{Fall not good in last <number> % of trail.}

BEGIN
   Cond29:= NOT( Cond28)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond30 : BOOLEAN;
{Rise slope - abs(fall slope) <= <number> * 0.01.}

BEGIN
    {Note - down B1 -ve, up +ve, so dif = up + down, forget abs.}
    Cond30:= ((Up_B1 + Down_B1) <= (Cond_In.C_Parms[1] * 0.01));
    IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Difference between rise and fall slopes was ',Up_B1 + Down_B1);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond31 : BOOLEAN;
{Inverse of cond 30.}

BEGIN
  Cond31:= ((Up_B1 + Down_B1) > (Cond_In.C_Parms[1] * 0.01))
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond32 : BOOLEAN;
{ABS( Straight line slope) <= <number> * 0.01.}

BEGIN
  Cond32:= (Absolute_Val( B1) <= (Cond_In.C_Parms[1] * 0.01));
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN('Straight line slope (for fit) was ',Absolute_Val( B1));
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond33 : BOOLEAN;
{ABS( Straight line slope) > <number> * 0.01.}

BEGIN
  Cond33:= (Absolute_Val( B1) > (Cond_In.C_Parms[1] * 0.01));
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond34 : BOOLEAN;
{Intersect of rise and fall > !<number> dBm above peak.}
   VAR X_Intersect, Y_Intersect : REAL;

BEGIN
  X_Intersect:= (Down_B0 - Up_B0) / ( Up_B1 - Down_B1);
  Y_Intersect:= Down_B0 + Down_B1 * X_Intersect;
  Cond34:= Y_Intersect > (High_Tide + Cond_In.C_Parms[1]);
  IF User_Wants_Actual_Condition_Value THEN BEGIN
      WRITE('Rise/fall straight line fit intersect was ',Y_Intersect);
      WRITELN(' dBm above true peak.');
  END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond35 : BOOLEAN;
{Intersect of rise and fall <= !<number> dBm above peak.}

BEGIN
   Cond35:= NOT( Cond34)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond36   : BOOLEAN;

{ Number of line segments needed for rise with tolerance .... }
{ !<number> is greater than !<number>.                        }

BEGIN
  Fit_Lines_To_Trail (1, Risepos, Cond_In.C_Parms [1] * 0.1,
     Maxint);
  Cond36 := Number_Of_Lines > Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond37   : BOOLEAN;

{ Number of line segments needed for rise with tolerance .... }
{ !<number> is less than or equal to !<number>.               }

BEGIN
  Fit_Lines_To_Trail (1, Risepos, Cond_In.C_Parms [1] * 0.1,
     Cond_In.C_Parms [2]);
  Cond37 := Number_Of_Lines <= Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond38   : BOOLEAN;

{ Number of line segments needed over whole trail with tolerance .... }
{ !<number> is greater than !<number>.                                }

BEGIN
  Fit_Lines_To_Trail (1, Trail_Array [0], Cond_In.C_Parms [1] * 0.1,
     Maxint);
  Cond38 := Number_Of_Lines > Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond39   : BOOLEAN;

{ Number of line segments needed over whole trail with tolerance .... }
{ !<number> is less than or equal to !<number>.                       }

BEGIN
  Fit_Lines_To_Trail (1, Trail_Array [0], Cond_In.C_Parms [1] * 0.1,
     Cond_In.C_Parms [2]);
  Cond39 := Number_Of_Lines <= Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond40  : BOOLEAN;

{ Number of lines calculated above with gradient less than ... }
{ Abs (!<number> * 0.01) is greater than !<number>.             }

VAR  Line, Slow_Slopes      : INTEGER;
     Gradient_Cutoff        : REAL;

BEGIN
  Slow_Slopes := 0;
  Gradient_Cutoff := Cond_In.C_Parms [1] * 0.01;
  FOR Line := 1 TO Number_Of_Lines DO
     IF Abs (Lines [Line].B1) <= Gradient_Cutoff THEN
        Slow_Slopes := Slow_Slopes + 1;
  Cond40 := Slow_Slopes > Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN('Number of lines with gradient as set was ',Slow_Slopes);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond41  : BOOLEAN;

{ Number of lines calculated above with gradient less than ... }
{ Abs (!<number> * 0.01) is less than or equal to !<number>.   }

BEGIN
  Cond41 := NOT Cond40
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond42  : BOOLEAN;

{ Lines calculated above consistent with twin-type - }
{ rise fall, rise fall.                              }

BEGIN
  IF Number_Of_Lines > 4 THEN  Cond42 := FALSE
  ELSE
    Cond42 := (Lines [1].B1 >= 0.1) AND
              (Lines [3].B1 >= 0.1) AND
              (Lines [2].B1 >= -0.5) AND (Lines [2].B1 < 0.0) AND
              (Lines [4].B1 >= -0.5) AND (Lines [4].B1 < 0.0)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond43 : BOOLEAN;
{Absolute minimum in !<Xth> !<number> % of trail.}
    VAR Min_Place, Region_Size : REAL;
            Start_Reg, End_Reg : REAL;
                  Check_Region : INTEGER;

BEGIN
   Check_Region:= Cond_In.C_Parms[1] - 100;   {1st, 2nd, etc.}
   Region_Size:= (Trail_Array[ 0] * 5.0) * Cond_In.C_Parms[2] / 100.0;
   {Region = number % of trail size.}
   Min_Place:= Min_Pos * 5.0;
   Start_Reg:= (Check_Region - 1) * Region_Size;
   End_Reg:= Check_Region * Region_Size;
   IF ((Start_Reg <= Min_Place) AND (Min_Place < End_Reg)) THEN
        Cond43:= TRUE
   ELSE Cond43:= FALSE;
   IF User_Wants_Actual_Condition_Value THEN BEGIN
       WRITE('Absolue minimum occurs after ');
       WRITELN(((Min_Pos/Trail_Array[0]) * 100.0):1:2,'% of the trail');
   END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond44 : BOOLEAN;
{Absolute minimum NOT in !<Xth> !<number> % of trail.}

BEGIN
   Cond44:= NOT( Cond43)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond45 : BOOLEAN;
{Abs min occurs before peak.}

BEGIN
   Cond45:= (Min_Pos < Peak_Pos)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond46 : BOOLEAN;
{Abs min after peak.}

BEGIN
   Cond46:= (Min_Pos > Peak_Pos)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond47 : BOOLEAN;
{Absolute minimum far more pronouned than other minima.}
   VAR Dist_Away, Trail_Dur  : REAL;
           More_Pronounced : BOOLEAN;
                         I : INTEGER;

BEGIN
  More_Pronounced:= TRUE;

  { Don't take last min into account }
  FOR I:= 1 TO Num_Minima - 1 DO
      IF ((Minima[ I, Yval] - 10) < Low_Tide) THEN BEGIN
         {Now check if not just subsidiary spike - ie. if}
         {a reasonable distance away from absolute min.  }
         Dist_Away:= Abs (Minima[I,Xval] - Min_Pos) * 5.0;
         Trail_Dur:= Trail_Array[0] * 5.0;
         IF ((Dist_Away/Trail_Dur * 100.0) > 10.0) THEN {If 10% away.}
             More_Pronounced:= FALSE;
      END;
  Cond47:= More_Pronounced
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond48 : BOOLEAN;
{Absolute minimum less pronouned than other minima.}

BEGIN
  Cond48:= NOT( Cond47)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond49   : BOOLEAN;

{ Returns true if peak in the middle of an existing upper plateau }
{ containing greater than !<number> % of trail.                   }

VAR  Plateau_Tol, Pos    : INTEGER;
     Peak_Middle         : BOOLEAN;

BEGIN
  IF Cond19 THEN
   BEGIN
     Peak_Middle := FALSE;
     Plateau_Tol := ROUND ((Plateaus [Highest_Platt, Eplatt] -
        Plateaus [Highest_Platt, Splatt] + 1) / 4);
     FOR Pos := Plateaus [Highest_Platt, Splatt] + Plateau_Tol TO
        Plateaus [Highest_Platt, Eplatt] - Plateau_Tol DO
        IF (Trail_Array [Pos] - 140) = High_Tide THEN
           Peak_Middle := TRUE;
     Cond49 := Peak_Middle
   END
  ELSE
     Cond49 := FALSE
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond50    : BOOLEAN;

{ Returns true if peak not in the middle of an existing upper plateau }
{ containing greater than !<number> % of trail.                       }

VAR  Plateau_Tol, Pos        : INTEGER;
     Not_Peak_Middle         : BOOLEAN;

BEGIN
  IF Cond19 THEN
   BEGIN
     Not_Peak_Middle := FALSE;
     Plateau_Tol := ROUND ((Plateaus [Highest_Platt, Eplatt] -
        Plateaus [Highest_Platt, Splatt] + 1) / 4);
     FOR Pos := Plateaus [Highest_Platt, Splatt] TO
        Plateaus [Highest_Platt, Splatt] + Plateau_Tol DO
        IF (Trail_Array [Pos] - 140) = High_Tide THEN
           Not_Peak_Middle := TRUE;
     FOR Pos := Plateaus [Highest_Platt, Eplatt] - Plateau_Tol TO
        Plateaus [Highest_Platt, Eplatt] DO
        IF (Trail_Array [Pos] - 140) = High_Tide THEN
           Not_Peak_Middle := TRUE;
     Cond50 := Not_Peak_Middle
   END
  ELSE
     Cond50 := FALSE
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond51 : BOOLEAN;
{Variance from parabola is greater than !<number> * 0.1.}

BEGIN
   Stu_Parabola( 1, Trail_Array[0]);
   Cond51:= (Parab_Var > (Cond_In.C_Parms[1] * 0.1));
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Variance from parabola was ',Parab_Var:1:2);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond52 : BOOLEAN;
{Variance from parabola is less than !<number> * 0.1.}

BEGIN
   Stu_Parabola( 1, Trail_Array[0]);
   Cond52:= (Parab_Var <= (Cond_In.C_Parms[1] * 0.1));
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Variance from parabola was ',Parab_Var:1:2);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond53 : BOOLEAN;
{Greater than <number> % of parabolas between mimima have variance}
{less than <number> * 0.1.                                         }
   VAR Good_Parbs, I : INTEGER;

BEGIN
  Good_Parbs:= 0;
  IF (Num_Minima <= 1) THEN
      Cond53:= FALSE
  ELSE BEGIN
     FOR I:= 1 TO (Num_Minima - 1) DO BEGIN
        Stu_Parabola( Minima[I,Xval], Minima[I+1,Xval]);
        IF (Parab_Var <= (Cond_In.C_Parms[2] * 0.1)) THEN
           Good_Parbs:= Good_Parbs + 1
     END;
     Cond53:= ((Good_Parbs/(Num_Minima - 1) * 100.0) > Cond_In.C_Parms[1])
  END;
  IF User_Wants_Actual_Condition_Value THEN
     WRITELN(Good_Parbs,' out of ',(Num_Minima - 1),' parabolas satisfied.');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond54 : BOOLEAN;
{Inverse of Cond53.}

BEGIN
   Cond54:= NOT( Cond53)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond55 : BOOLEAN;
{There are more than !<number> local extrema.}

BEGIN
   Cond55:= (Num_Extrema > Cond_In.C_Parms[1])
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond56 : BOOLEAN;
{There are less than !<number> local extrema.}

BEGIN
   Cond56:= (Num_Extrema <= Cond_In.C_Parms[1]);
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('There were ',Num_Extrema,' extremas.');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond57 : BOOLEAN;
{Time since last sample is greater than !<number> * 0.1 seconds.}
   VAR Time_Gap : REAL;

BEGIN
   Time_Gap:= One_Time( Hrec.Start_Time) -
              (One_Time( Prev_Rec.Start_Time) +
               (Prev_Rec.Num_Samples * 0.5));
   IF (Time_Gap = 0.0) THEN    {First record in file.}
        Cond57:= TRUE
   ELSE Cond57:= Time_Gap/10.0 > Cond_In.C_Parms[1];
   {Time gap in hundredths, need div 10 for tenths.}
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Time since last trail is ',(Time_Gap/100.0):1:2,' seconds.');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond58 : BOOLEAN;
{Time since last sample is less or equal to !<number> * 0.1 seconds.}
   VAR Time_Gap : REAL;

BEGIN
   Time_Gap:= One_Time( Hrec.Start_Time) -
             (One_Time( Prev_Rec.Start_Time) +
               (Prev_Rec.Num_Samples * 0.5));
   IF (Time_Gap = 0.0) THEN    {First record in file.}
        Cond58:= FALSE
   ELSE Cond58:= Time_Gap/10.0 <= Cond_In.C_Parms[1]
   {Time gap in hundredths, need div 10 for tenths.}
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond59 : BOOLEAN;
{Last sample mean within !<number> dBm of current sample mean.}
   VAR Meandif : REAL;

BEGIN
   Meandif:= Absolute_Val( Hrec.Sample_Mean - Prev_Rec.Sample_Mean);
   Cond59:= (Meandif <= Cond_In.C_Parms[1]);
   IF User_Wants_Actual_Condition_Value THEN BEGIN
      WRITE('Difference between last & current sample means is ');
      WRITELN( Meandif,'.');
   END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond60 : BOOLEAN;
{Last sample mean not within !<number> dBm of current sample mean.}

BEGIN
  Cond60:= NOT( Cond59)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond61 : BOOLEAN;
{Checks if previous trail duration greater than <number> times 50 mSec.}
   VAR Duration_On_Sample : REAL;

BEGIN
    Duration_On_Sample:= Prev_Rec.Num_Samples * 5;  {Num samples * 5}
    IF (Duration_On_Sample > (Cond_In.C_Parms[1] * 50.0)) THEN
         Cond61:= TRUE
    ELSE Cond61:= FALSE;
    IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Duration of previous trail was ',Duration_On_Sample,' ms');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond62: BOOLEAN;
{Checks if previous trail duration less or equal to <number> times 50 mSec.}

BEGIN
   Cond62:= NOT (Cond61)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond63 : BOOLEAN;
{Greater than <number> % of parabolas between maxima have variance}
{less than <number> * 0.1.                                         }
   VAR Good_Parbs, I : INTEGER;

BEGIN
  Good_Parbs:= 0;
  IF (Num_Maxima <= 1) THEN Cond63:= FALSE
  ELSE BEGIN
    FOR I:= 1 TO (Num_Maxima - 1) DO BEGIN
        Stu_Parabola( Maxima[I,Xval], Maxima[I+1,Xval]);
        IF (Parab_Var <= (Cond_In.C_Parms[2] * 0.1)) THEN
            Good_Parbs:= Good_Parbs + 1
    END;
    Cond63:= ((Good_Parbs/(Num_Maxima - 1) * 100.0) > Cond_In.C_Parms[1])
  END;
  IF User_Wants_Actual_Condition_Value THEN BEGIN
       WRITE( (Good_Parbs/(Num_Maxima - 1) * 100.0):1:2,'% of parabolas');
       WRITELN(' have variance within the tolerance.');
  END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond64 : BOOLEAN;
{Inverse of Cond63.}

BEGIN
   Cond64:= NOT( Cond63)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond65 : BOOLEAN;
{One good parabola fit (variance <= <number> * 0.1), gets 50% of trail.}
   VAR Gotbiggie, I : INTEGER;

BEGIN
   Gotbiggie:= -1;
   FOR I:= 1 TO Num_Minima - 1 DO
       IF ((Minima[I + 1, Xval] - Minima[ I, Xval] + 1) > Trail_Array[0] DIV 2)
          THEN
              Gotbiggie:= I;
   IF (Gotbiggie <> -1) THEN BEGIN
       Stu_Parabola( Minima[ Gotbiggie, Xval], Minima[ (Gotbiggie+1), Xval]);
       Cond65:= (Parab_Var <= (Cond_In.C_Parms[1] * 0.1))
   END
   ELSE Cond65:= FALSE;
   IF User_Wants_Actual_Condition_Value THEN
       IF (Gotbiggie = -1) THEN
            WRITELN('No parabola fit covers 50% of the trail.')
       ELSE WRITELN('Varaince on the parabola fit was ',Parab_Var:1:2);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond66 : BOOLEAN;
{Inverse of Cond65}

BEGIN
   Cond66:= NOT (Cond65)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond67 : BOOLEAN;
{Checks if fall variance from straight line greater than <number> * 0.1.}

BEGIN
   Cond67:= (Down_Variance > (Cond_In.C_Parms[1] * 0.1));
   IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Fall variance was ',Down_Variance:1:2);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond68 : BOOLEAN;
{Checks if fall variance <= <number * 0.1.}

BEGIN
   Cond68:= NOT( Cond67)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond69 : BOOLEAN;
{Checks if rise variance from straight line greater than <number> * 0.1.}

BEGIN
   Cond69:= (Up_Variance > (Cond_In.C_Parms[1] * 0.1));
   IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Rise variance was ',Up_Variance);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond70 : BOOLEAN;
{Checks if rise variance <= <number * 0.1.}

BEGIN
   Cond70:= NOT( Cond69)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond71 : BOOLEAN;
{Fall longer than upper plateau.}
    VAR Plat_Div_5, Fall_Div_5 : INTEGER;

BEGIN
   Plat_Div_5:= Plateaus[ Highest_Platt, Eplatt] -
                Plateaus[ Highest_Platt, Splatt] + 1;
   Fall_Div_5:= Trail_Array[0] - Plateaus[ Highest_Platt, Eplatt] + 1;
   Cond71:= (Fall_Div_5 > Plat_Div_5);
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Fall : ',Fall_Div_5 * 5,'  Plateau : ',Plat_Div_5 * 5,' (ms)');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond72 : BOOLEAN;

BEGIN
  Cond72:= NOT( Cond71)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}



{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond73   : BOOLEAN;

{ Number of line segments needed over fall with tolerance .... }
{ !<number> is greater than !<number>.                         }

BEGIN
  Fit_Lines_To_Trail (Fallpos, Trail_Array [0],
     Cond_In.C_Parms [1] * 0.1, Maxint);
  Cond73 := Number_Of_Lines > Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond74   : BOOLEAN;

{ Number of line segments needed over fall with tolerance .... }
{ !<number> is less than or equal to !<number>.                }

BEGIN
  Fit_Lines_To_Trail (Fallpos, Trail_Array [0],
     Cond_In.C_Parms [1] * 0.1, Cond_In.C_Parms [2]);
  Cond74 := Number_Of_Lines <= Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
       WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond75   : BOOLEAN;

{ Number of line segments needed over whole trail with tolerance .... }
{ !<number> is equal to !<number>.                                    }

BEGIN
  Fit_Lines_To_Trail (1, Trail_Array [0],
     Cond_In.C_Parms [1] * 0.1, Cond_In.C_Parms [2]);
  Cond75 := Number_Of_Lines = Cond_In.C_Parms [2];
  IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Number of segments needed was ',Number_Of_Lines);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond76   : BOOLEAN;

{ Line segments calculated above rise for more than !<number> % of fall. }

CONST  Rise_Tolerance = 0.0;

VAR  Fall_Length, Rise_Length, Max_Rise_Length, Line      : INTEGER;
     Rising                                               : BOOLEAN;

BEGIN
  Fall_Length := Trail_Array [0] - Fallpos + 1;
  Max_Rise_Length := 0;
  Line := 1;
  WHILE Line <= Number_Of_Lines DO
   BEGIN
     Rise_Length := 0;
     Rising := Lines [Line].B1 > Rise_Tolerance;
     WHILE (Rising) AND (Line <= Number_Of_Lines) DO
      BEGIN
        Rise_Length := Rise_Length + Lines [Line].Stop -
                       Lines [Line].Start + 1;
        Line := Line + 1;
        Rising := Lines [Line].B1 > Rise_Tolerance
      END;
     IF Rise_Length > Max_Rise_Length THEN
        Max_Rise_Length := Rise_Length;
     Line := Line + 1
   END;
  Cond76 := (Max_Rise_Length / Fall_Length * 100.0) > Cond_In.C_Parms [1];
  IF User_Wants_Actual_Condition_Value THEN BEGIN
    WRITE('% of fall where line fit rising is ');
    WRITELN( ((Max_Rise_Length/Fall_Length) * 100.0):1:2,'%');
  END;
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond77   : BOOLEAN;

{ Line segments calculated above rise for less than or equal to }
{ !<number> % of fall.                                          }

BEGIN
  Cond77 := NOT Cond76
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond78 : BOOLEAN;
{Rise duration is greater than !<number> * 5 mSec.}
    VAR Rise_Dur : INTEGER;

BEGIN
   Rise_Dur:= Risepos - 1;
   Cond78:= (Rise_Dur > Cond_In.C_Parms[1]);
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Rise duration was ',Rise_Dur * 5,' milliseconds.');
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond79 : BOOLEAN;

BEGIN
   Cond79:= NOT( Cond78)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond80 : BOOLEAN;
{Curve coefficient of last parabola calculated greater than !<number> - 80.}
    VAR Curve_Coeff : INTEGER;

BEGIN
   Curve_Coeff:= ROUND( A_Coeff * Trail_Array[0] * Trail_Array[0]);
   Cond80:= (Curve_Coeff > (Cond_In.C_Parms[1] - 80));
   IF User_Wants_Actual_Condition_Value THEN
      WRITELN('Curve coefficient of last parabola was ',Curve_Coeff);
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond81 : BOOLEAN;
{Not Cond80.}

BEGIN
   Cond81:= NOT( Cond80)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}


{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond82 : BOOLEAN;
{Good parabola fit from most pronounced local minimum on fall.}
   VAR Curving_Coeff, Dif : INTEGER;

BEGIN
  IF (Worst_Line_Dif_Pos = Trail_Array[0]) THEN
       Cond82:= FALSE
  ELSE BEGIN
       Stu_Parabola( Worst_Line_Dif_Pos, Trail_Array[0]);
       IF (Parab_Var > 1.2) THEN
            Cond82:= FALSE
       ELSE BEGIN
            Dif:= Trail_Array[0] - Worst_Line_Dif_Pos + 1;
            Curving_Coeff:= ROUND( A_Coeff * Dif * Dif);
            Cond82:= (Curving_Coeff <= -5)
       END
  END
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
```

```
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
FUNCTION Cond83 : BOOLEAN;
{Not a good parabola fit from most pronounced minimum on fall.}

BEGIN
   Cond83:= NOT( Cond82)
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}




BEGIN
   Cnd_Num:= Cond_In.Cond_Num;
   CASE Cnd_Num OF
        1 : Condition_Checker:= Cond1;
        2 : Condition_Checker:= Cond2;
        3 : Condition_Checker:= Cond3;
        4 : Condition_Checker:= Cond4;
        5 : Condition_Checker:= Cond5;
        6 : Condition_Checker:= Cond6;
        7 : Condition_Checker:= Cond7;
        8 : Condition_Checker:= Cond8;
        9 : Condition_Checker:= Cond9;
       10 : Condition_Checker:= Cond10;
       11 : Condition_Checker:= Cond11;
       12 : Condition_Checker:= Cond12;
       13 : Condition_Checker:= Cond13;
       14 : Condition_Checker:= Cond14;
       15 : Condition_Checker:= Cond15;
       16 : Condition_Checker:= Cond16;
       17 : Condition_Checker:= Cond17;
       18 : Condition_Checker:= Cond18;
       19 : Condition_Checker:= Cond19;
       20 : Condition_Checker:= Cond20;
       21 : Condition_Checker:= Cond21;
       22 : Condition_Checker:= Cond22;
       23 : Condition_Checker:= Cond23;
       24 : Condition_Checker:= Cond24;
       25 : Condition_Checker:= Cond25;
       26 : Condition_Checker:= Cond26;
       27 : Condition_Checker:= Cond27;
       28 : Condition_Checker:= Cond28;
       29 : Condition_Checker:= Cond29;
       30 : Condition_Checker:= Cond30;
       31 : Condition_Checker:= Cond31;
       32 : Condition_Checker:= Cond32;
       33 : Condition_Checker:= Cond33;
       34 : Condition_Checker:= Cond34;
       35 : Condition_Checker:= Cond35;
       36 : Condition_Checker:= Cond36;
       37 : Condition_Checker:= Cond37;
       38 : Condition_Checker:= Cond38;
       39 : Condition_Checker:= Cond39;
       40 : Condition_Checker:= Cond40;
       41 : Condition_Checker:= Cond41;
       42 : Condition_Checker:= Cond42;
       43 : Condition_Checker:= Cond43;
       44 : Condition_Checker:= Cond44;
       45 : Condition_Checker:= Cond45;
       46 : Condition_Checker:= Cond46;
       47 : Condition_Checker:= Cond47;
       48 : Condition_Checker:= Cond48;
       49 : Condition_Checker:= Cond49;
       50 : Condition_Checker:= Cond50;
       51 : Condition_Checker:= Cond51;
       52 : Condition_Checker:= Cond52;
       53 : Condition_Checker:= Cond53;
       54 : Condition_Checker:= Cond54;
       55 : Condition_Checker:= Cond55;
       56 : Condition_Checker:= Cond56;
       57 : Condition_Checker:= Cond57;
```

```
      58 : Condition_Checker:= Cond58;
      59 : Condition_Checker:= Cond59;
      60 : Condition_Checker:= Cond60;
      61 : Condition_Checker:= Cond61;
      62 : Condition_Checker:= Cond62;
      63 : Condition_Checker:= Cond63;
      64 : Condition_Checker:= Cond64;
      65 : Condition_Checker:= Cond65;
      66 : Condition_Checker:= Cond66;
      67 : Condition_Checker:= Cond67;
      68 : Condition_Checker:= Cond68;
      69 : Condition_Checker:= Cond69;
      70 : Condition_Checker:= Cond70;
      71 : Condition_Checker:= Cond71;
      72 : Condition_Checker:= Cond72;
      73 : Condition_Checker:= Cond73;
      74 : Condition_Checker:= Cond74;
      75 : Condition_Checker:= Cond75;
      76 : Condition_Checker:= Cond76;
      77 : Condition_Checker:= Cond77;
      78 : Condition_Checker:= Cond78;
      79 : Condition_Checker:= Cond79;
      80 : Condition_Checker:= Cond80;
      81 : Condition_Checker:= Cond81;
      82 : Condition_Checker:= Cond82;
      83 : Condition_Checker:= Cond83;
  END
END;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}

END.
```

```
UNIT Ts4;
{This unit contains the initialisation and the finalisation routines}

Interface

USES Dos, Crt, Graph, Ts1;

PROCEDURE Initialise;
PROCEDURE Finalise;


IMPLEMENTATION


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Initialise;

CONST Blank_Inside = TRUE;
      Leave_Inside = FALSE;

      Play = TRUE;
       Bid = FALSE;


TYPE Str3 = String[3];


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Init_Graphics;

BEGIN
   Graphdriver:= Detect;
   Initgraph( Graphdriver, Graphmode, 'C:DRIVERS');
   Errorcode:= Graphresult;
   IF (Errorcode <> Grok) THEN BEGIN
      WRITELN('Graphics problem - ',Grapherrormsg( Errorcode));
      WRITELN('Program aborted');
      Halt(1);
   END
   ELSE BEGIN
      Settextstyle(Smallfont, Vertdir, 6);
      Restorecrtmode
   END
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Horizontal_Line( X_Start, X_End, Y_Point : INTEGER;
                          Str_In : Str3; Blank_Out : BOOLEAN);

   VAR Left_Char, Right_Char, Mid_Char : CHAR;
                              X_Point : INTEGER;

BEGIN
  Left_Char:= Str_In[1];
  Mid_Char:= Str_In[2];
  Right_Char:= Str_In[3];
  Gotoxy( X_Start, Y_Point);
  WRITE( Left_Char);
  IF ((Mid_Char <> ' ') OR Blank_Out) THEN
      FOR X_Point:= (X_Start + 1) TO (X_End - 1) DO
          WRITE( Mid_Char);
  Gotoxy( X_End, Y_Point);
  WRITE( Right_Char);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{------------------------------------------------------------------}
PROCEDURE Draw_Margin( X_Start, X_End, Y_Start, Y_End : INTEGER;
                       Upper_Str, Mid_Str, Lower_Str : Str3;
                                           Blank_Out : BOOLEAN);

   VAR Y_Point : INTEGER;

BEGIN
  Horizontal_Line( X_Start, X_End, Y_Start, Upper_Str, Blank_Out);
  FOR Y_Point:= (Y_Start + 1) TO (Y_End - 1) DO
       Horizontal_Line( X_Start, X_End, Y_Point, Mid_Str, Blank_Out);
  Horizontal_Line( X_Start, X_End, Y_End, Lower_Str, Blank_Out)
END;
{------------------------------------------------------------------}



{------------------------------------------------------------------}
PROCEDURE Do_Main_Margins;

VAR Top_Inv, Mid_Inv, Bot_Inv, Top_Dub, Mid_Dub, Bot_Dub : Str3;
       Margin_Char, I : INTEGER;
BEGIN
   Margin_Char:= 219;
   Top_Inv:= CHR(Margin_Char) + CHR(Margin_Char) + CHR(Margin_Char);
   Mid_Inv:= CHR(Margin_Char) + ' ' + CHR(Margin_Char);
   Bot_Inv:= CHR(Margin_Char) + CHR(Margin_Char) + CHR(Margin_Char);
   Top_Dub:= CHR(201) + CHR(205) + CHR(187);
   Mid_Dub:= CHR(186) + ' ' + CHR(186);
   Bot_Dub:= CHR(200) + CHR(205) + CHR(188);
   Clrscr;
   Draw_Margin( 10, 66, 3, 9, Top_Dub, Mid_Dub, Bot_Dub, Leave_Inside);
   Draw_Margin( 10, 66, 12, 23, Top_Inv, Mid_Inv, Bot_Inv, Leave_Inside);
END;
{------------------------------------------------------------------}



{------------------------------------------------------------------}
PROCEDURE Write_Team( Offset, St_Row : INTEGER);

CONST Happy = #2;

BEGIN
  Normvideo;
  St_Row:= St_Row + 2;
  Gotoxy( Offset, St_Row + 8);
  WRITE('             THE  TRAILSTAR  TEAM');
  Gotoxy( Offset, St_Row + 9);
  WRITE('             ~~~~~~~~~~~~~~~~~~~~');
  Gotoxy( Offset, St_Row + 10);
  Lowvideo;
  WRITE( '  ', Happy, '  Expert system design by : ');
  Gotoxy( Offset, St_Row + 11);
  WRITE( '  ', Happy, '  Graphics and display by : ');
  Gotoxy( Offset, St_Row + 12);
  WRITE( '  ', Happy, '  Mathematical consultant : ');
  Gotoxy( Offset, St_Row + 13);
  WRITE( '  ', Happy, '  Original system  expert : ');
  Normvideo;
  Offset:= Offset + 31;
  Gotoxy( Offset, St_Row + 10);
  WRITE('Stuart Melville');
  Gotoxy( Offset, St_Row + 11);
  WRITE('Robert Letschert');
  Gotoxy( Offset, St_Row + 12);
  WRITE('Wayne Goddard');
  Gotoxy( Offset, St_Row + 13);
  WRITE('James Larsen');
  Gotoxy( Offset - 30, St_Row + 15);
  WRITE('Programmed by Stuart Melville and Robert Letschert');
END;
{------------------------------------------------------------------}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Show_Author_Version;
   CONST Offset = 12;
         St_Row = 4;

BEGIN
  Gotoxy(Offset, St_Row);
  Normvideo;
  WRITE('TrailStar  -  a rule-based expert system which types');
  Gotoxy(Offset, St_Row + 1);
  WRITE('and learns to type meteor trails.');
  Gotoxy(Offset, St_Row + 3);
  WRITE(' Adapted from the BridgeStar system. (Melville 1986)');
  Gotoxy(Offset, St_Row + 4);
  Lowvideo;
  WRITE(' (Version  ');
  WRITE('4.04    ');
  WRITE('   Last modified : ');
  WRITE('July 1989');
  Lowvideo;
  WRITE(')');
  Write_Team( Offset, St_Row);
END;
{-----------------------------------------------------------------------}



{-----------------------------------------------------------------------}
PROCEDURE Go_Write( Xpos, Ypos : INTEGER; Ch : CHAR);

BEGIN
   Gotoxy( Xpos, Ypos);
   WRITE( Ch)
END;
{-----------------------------------------------------------------------}



{-----------------------------------------------------------------------}
PROCEDURE Give_Message_And_Hide_Cursor;

BEGIN
  Gotoxy(1,25);
  WRITE('        ');
  WRITE(CHR(1));
  WRITE('         Waking up system - with you in a moment...');
  Gotoxy( 10, 12);
END;
{-----------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Play_Music;

CONST Short = 125;
      Long = 375;
      Mid = 250;
      Vlong = 500;
      Vshort = 75;


BEGIN
(*
  Sound( 523);
  Delay( Vlong);
  Nosound;
  Delay( Vshort);
  Sound( 786);
  Delay( Long);
  Nosound;
  Delay( Vshort);
  Sound( 659);
  Delay(Short);
  Nosound;
  Delay(Vshort);
  Sound( 698);
  Delay( Vlong);
  Nosound  *)
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Show_Logo;

BEGIN
  Do_Main_Margins;
  Show_Author_Version;
  Give_Message_And_Hide_Cursor;
  IF Sound_Flag THEN
     Play_Music;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Set_Sets;
{Initialises all the global sets used.}
   VAR I : BYTE;

BEGIN
  Yesno:= ['Y','N'];
  Spaces:= [' ',',','.',';'];
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Set_Global_Vars;

BEGIN
  Files_Ok:= TRUE;
  Exitting:= FALSE;
  Choice:= 'A';
  Curname:= 'UNKNOWN';
  Basenum:= 0;
  Bases_Available:= [];
  Curbase:= 'No current base';
  User_Wants_Actual_Condition_Value:= FALSE;
  Number_Of_Lines := 0
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Open_Files;
{Open all files on logged drive to be used, and the printer file.}
    VAR Present_Check : ARRAY[ 1..4] OF BOOLEAN;
               All_Ok : BOOLEAN;



PROCEDURE Write_Attendance( Which_One : INTEGER);

BEGIN
   IF (Present_Check[ Which_One] = TRUE) THEN
       WRITELN('   -  present.')
   ELSE WRITELN('   -  not present.')
END;
```

```
{-------------------------------------------------------------------------}
PROCEDURE File_Error;
{Alerts user to file error, and informs as to source of error.}

BEGIN
   Alarm( Leave_Cursor);
   IF (Files_Ok) THEN BEGIN      {If first file error found.}
      Files_Ok:= FALSE;
      Clrscr
   END;
   Normvideo;
   Gotoxy(1,1);
   WRITELN('             !!!!  FILE ERROR  !!!');
   WRITELN;
   WRITELN('** - Unable to find vital file(s) on the logged drive.');
   WRITELN('All of the  following files are needed on  the  logged');
   WRITELN('drive if the program is to be  correctly run (a ''?'' in');
   WRITELN('the file name indicates some number -  if a RULES file');
   WRITELN('has a certain number, there MUST be a  matching  PRIOR');
   WRITELN('file) : ');
   WRITE('        BASECNDS.STA');
   Write_Attendance(1);
   WRITE('        BASEACTS.STA');
   Write_Attendance(2);
   WRITE('        RULES?.STA');
   Write_Attendance(3);
   WRITE('        PRIORIT?.STA');
   Write_Attendance(4);
   WRITELN;
   WRITELN;
   WRITELN('Note - alternate source of error is that your');
   WRITELN('operating system has too  few file buffers - ');
   WRITELN('ensure FILES = 20 set in your CONFIG.SYS.');
   WRITELN;
   Pause_It
END;
{-------------------------------------------------------------------------}



{-------------------------------------------------------------------------}
FUNCTION Reset_Check( Fnam : Str20; Fnum : INTEGER) : BOOLEAN;

BEGIN
    {$I-}
    CASE Fnum OF
         1 : BEGIN
                ASSIGN( Cbase, Fnam);
                RESET( Cbase)
             END;
         2 : BEGIN
                ASSIGN( Abase, Fnam);
                RESET( Abase)
             END;
    END;
    IF (Ioresult = 0) THEN
         Reset_Check:= TRUE
    ELSE BEGIN
         All_Ok:= FALSE;
         Reset_Check:= FALSE
    END
    {$I+}
END;
{-------------------------------------------------------------------------}
```

```
{------------------------------------------------------------------------}
FUNCTION Extract_Num_Part( Nam : String) : INTEGER;
  VAR Pos_In_String, Err, Numval : INTEGER;
                        Numpart : String;

BEGIN
    Numpart:= '';
    Pos_In_String:= 6;
    REPEAT
       IF (Nam[ Pos_In_String] <> '.') THEN BEGIN
              Numpart:= Numpart + Nam[ Pos_In_String];
              Pos_In_String:= Pos_In_String + 1;
       END;
    UNTIL (Nam[ Pos_In_String] = '.');
    VAL( Numpart, Numval, Err);
    Extract_Num_Part:= Numval
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
FUNCTION Presence_Check( Fnam : Str20; Fnum : INTEGER) : BOOLEAN;
  VAR Dirinfo : Searchrec;
        Match : BOOLEAN;
         Pnam : String;
          Pos : INTEGER;

BEGIN
    IF (Fnum = 3) THEN BEGIN     {Find available bases}
       Findfirst( 'RULES*.STA', Anyfile, Dirinfo);
       WHILE (Doserror = 0) DO BEGIN
             Basenum:= Extract_Num_Part( Dirinfo.Name);
             Bases_Available:= Bases_Available + [Basenum];
             Findnext( Dirinfo);
       END;
       All_Ok:= (Bases_Available <> []);
       Presence_Check:= All_Ok;
    END
    ELSE BEGIN
         Match:= TRUE;
         FOR Pos:= 1 TO Topbase DO BEGIN
             IF (Pos IN Bases_Available) THEN BEGIN
                 STR( Pos, Pnam);
                 Pnam:= 'PRIOR' + Pnam + '.STA';
                 Findfirst( Pnam, Anyfile, Dirinfo);
                 IF (Doserror <> 0) THEN
                     Match:= FALSE;
             END
         END;
         All_Ok:= Match;
         Presence_Check:= All_Ok
    END;
END;
{------------------------------------------------------------------------}


BEGIN                             {Open files routine.}
    All_Ok:= TRUE;
    Present_Check[1]:= Reset_Check('BASECNDS.STA', 1);
    Present_Check[2]:= Reset_Check('BASEACTS.STA', 2);
    Present_Check[3]:= Presence_Check('RULES.STA', 3);
    Present_Check[4]:= Presence_Check('PRIOR.STA', 4);
    ASSIGN( Prt, 'LPT1');
    REWRITE( Prt);
    IF (NOT All_Ok) THEN
         File_Error;
END;
{------------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Say_Ready;

BEGIN
  WRITE(CHR(7));
  Gotoxy(1,25);
  WRITE('                    ');
  WRITE(CHR(2));
  WRITE(' All Well! ');
  WRITE(CHR(2));
  WRITE('     Press any key to start...   ');
  Gotoxy( 10, 12);
  Anykey:= Readkey;
  IF (Anykey = #27) THEN
       Debug_Flag:= TRUE
  ELSE Debug_Flag:= FALSE;
  Lowvideo;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{Beginning of initialisation procedure.}


BEGIN
   Init_Graphics;
   Show_Logo;
   Set_Sets;
   Set_Global_Vars;
   Open_Files;
   Basenum:= 0;
   Curbase:= 'No current base';
   IF Files_Ok THEN
      Say_Ready;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}




{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Finalise;

BEGIN
   CLOSE( Abase);
   CLOSE( Cbase);
   CLOSE( Prt);
   Clrscr;
   Gotoxy(15,12);
   Normvideo;
   WRITE('All done for now - have a good day!');
   Delay(1200);
   Lowvideo;
   Gotoxy(18,24);
   WRITE('Herendeth the session.');
   Closegraph
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


END.
```

```
UNIT Ts5;
{This unit is implemented as an aid to high-level modifiers of}
{TrailStar.  It allows for the addition of new  text  for  the}
{condition and action bases to describe and act  as  parameter}
{specification for  any  new  condition-check/action-determine}
{routines the modifier adds, and also allows for change in old}
{condition/action text.                                       }

Interface

USES Ts1, Dos, Crt;

PROCEDURE High_Level_Modifier;


IMPLEMENTATION


{#####################################################################}
PROCEDURE High_Level_Modifier;


CONST Actbase = 1;
      Cndbase = 2;

VAR Basenam : ARRAY[1..2] OF Str20;


{#####################################################################}
PROCEDURE Warn_Danger;

BEGIN
   Menu_Head;
   WRITELN('                    HIGH  LEVEL  MODIFICATION.');
   WRITELN;
   WRITELN('    !!!!  WARNING  !!!!');
   WRITELN;
   Lowvideo;
   WRITELN('If you are not a  skilled computer programmer then you should');
   WRITELN('not be here! Press 'X' to exit immediately.');
   WRITELN;
   WRITELN('If you ARE a skilled programmer, but are not fully conversant');
   WRITELN('with "The  High-Level Modifier's Guide to TrailStar" then the');
   WRITELN('same applies - can't learn modification on the fly.');
   WRITELN;
   WRITELN('If you intend to carry on, and you haven't got copies of  the');
   WRITELN('old condition/action files backed up somewhere then you are :');
   WRITELN('      (a) an idiot;');
   WRITELN('      (b) a braver man than myself;');
   WRITELN('  or  (c) both.');
   Lowvideo
END;
{#####################################################################}



{#####################################################################}
FUNCTION Knows_Way_In : BOOLEAN;
{Checks whether user knows the 'pass-character'.}
   VAR Char_Chosen : CHAR;

BEGIN
   Gotoxy(1,24);
   WRITE('Please enter decision (any char bar code char exits) : ');
   Char_Chosen:= Readkey;
   Clrscr;
   IF (Char_Chosen = 'F') THEN
        Knows_Way_In:= TRUE
   ELSE Knows_Way_In:= FALSE
END;
{#####################################################################}
```

```
{#####################################################################}
PROCEDURE Ask_Type_Of_Modif( VAR Change_Or_New : CHAR);

BEGIN
   Menu_Head;
   Lowvideo;
   Gotoxy(1,14);
   WRITELN('You have two options here. Enter 'C' if you wish to  change');
   WRITELN('the current text for  some  condition/action/indication, or');
   WRITELN(''N' if you wish to add new conditions/actions/indications.');
   WRITELN;
   WRITE('Please enter choice : ');
   Change_Or_New:= Get_Valid_Choice_From(['C','N']);
END;
{#####################################################################}




{#####################################################################}
PROCEDURE Stress_Operations_To_Perform;

BEGIN
   Menu_Head;
   Lowvideo;
   WRITELN('The high-level modifier's role is to create new condition-check');
   WRITELN('and action-determining routines. As these conditions  & actions');
   WRITELN('will be used in rules created by a user, an onus falls  on  you');
   WRITELN('to make them idiot-proof. While an in-built consistency checker');
   WRITELN('will pick up plays that are patently garbage and report  to the');
   WRITELN('user on the rule-base flaw, there is no way it can know  that a');
   WRITELN('condition actually checks what it purports to,  or if an action');
   WRITELN('returns the typing it was supposed to, since these routines are');
   WRITELN('written by high-level modifiers.');
   WRITELN;
   WRITELN('Take especial care with parameters - when the user decides on a');
   WRITELN('cond/act for a rule, he/she can choose any  parameters  desired');
   WRITELN('which is acceptable in parameter context.');
   WRITELN;
   WRITELN('Remember to write up action & condition routines in  files  TS3');
   WRITELN('and TS4 respectively, and to make an entry for the new  routine');
   WRITELN('in the CASE statement at beginning of file, and TEST thoroughly');
   WRITELN('before release to users - else on your head be the crash!');
   WRITELN;
   WRITE('Any key to continue.');
   Anykey:= Readkey
END;
{#####################################################################}



{#####################################################################}
FUNCTION Wants_To_Carry_On : BOOLEAN;
{Checks if user wants to add more new conditions/actions.}
   VAR Go_On : CHAR;

BEGIN
   Lowvideo;
   Gotoxy(1,24);
   WRITE('Enter 'Q' to Quit text entry, 'C' to continue.');
   Go_On:= Get_Valid_Choice_From(['Q','C']);
   IF (Go_On = 'C') THEN
        Wants_To_Carry_On:= TRUE
   ELSE Wants_To_Carry_On:= FALSE
END;
{#####################################################################}
```

```
{#####################################################################}
PROCEDURE Store( Base : BYTE; Fplace : INTEGER; Txt_Str : Str80);
{Saves text to appropriate base at position FPLACE. This position}
{will be at end of file in the case of new actions/conditions.    }

BEGIN
   IF (Base = Cndbase) THEN BEGIN
        SEEK( Cbase, Fplace);                  {Seek to EOF.        }
        WRITE( Cbase, Txt_Str);
        CLOSE( Cbase);                         {Ensure change saved.}
        RESET( Cbase)
   END
   ELSE BEGIN                                  {ActBase.            }
        SEEK( Abase, Fplace);                  {Seek to EOF.        }
        WRITE( Abase, Txt_Str);
        CLOSE( Abase);                         {Ensure change saved.}
        RESET( Abase)
   END
END;
{#####################################################################}



{#####################################################################}
PROCEDURE Get_And_Enter_Text( Base : BYTE;  Fplace : INTEGER);
{Tells the user how text and parameter flags within text should}
{be entered, gets text, and calls STORE to place in appropriate}
{base.                                                          }
    VAR Txt_Str : Str80;

BEGIN
  Menu_Head;
  WRITE('Text entry for file ');
  WRITE( Basenam[ Base]);
  Lowvideo;
  WRITELN;
  WRITELN('REMEMBER : The '!' symbol has a special role in that it is seen as');
  WRITELN('showing a parameter by both system and user.  It should always be');
  WRITELN('followed by a valid parameter type - eg. '<number>','<Xth>', etc.');
  WRITELN;
  WRITELN('Apart from '!' symbols, all text is ignored by system - all system');
  WRITELN('concerned with is that text there => cond/act exists, and text can');
  WRITELN('be  shown  to user, with prompts for parms on rule creation/alter.');
  WRITELN('Thus clarity your only criterion. To maintain consistency  in  the');
  WRITELN('presentation  of  conditions,  write text in statement form rather');
  WRITELN('than conditional. eg "Good rise" rather than "If good rise."');
  Gotoxy(1,23);
  Normvideo;
  WRITELN('Please enter text on next line, <RETURN> when done.');
  READLN( Txt_Str);
  Store( Base, Fplace, Txt_Str)
END;
{#####################################################################}
```

```
{###################################################################}
PROCEDURE Change_Text;
{Determines whether action or condition text that modifier wishes}
{to alter, and determines number of this action/condition (number)}
{maps directly onto file position). Repeatedly does this, calling}
{GET_AND_ENTER_TEXT to store new text, until user responds 'N' to}
{the "Change another (Y/N) ? " prompt.                            }
   VAR Change_Char : CHAR;
           Fplace : INTEGER;
             Base : BYTE;

BEGIN
   Menu_Head;
   WRITELN('           Text  Alteration.');
   Lowvideo;
   REPEAT
       Gotoxy(1,24);
       Clreol;
       WRITE('Change action or condition text (A/C) ? ');
       Change_Char:= Get_Valid_Choice_From(['A','C']);
       IF (Change_Char = 'A') THEN
           Base:= Actbase
       ELSE Base:= Cndbase;
       Gotoxy(1,24);
       Clreol;
       WRITE('Number of condition/action to change : ');
       READLN( Fplace);
       Get_And_Enter_Text( Base, (Fplace - 1));
       Gotoxy(1,24);
       Clreol;
       WRITE('Change another (Y/N) ? ');
       Change_Char:= Get_Valid_Choice_From( Yesno)
   UNTIL (Change_Char = 'N')
END;
{###################################################################}



{###################################################################}
PROCEDURE Determine_Size_And_Entity( VAR Fsize : INTEGER;
                VAR Entity : Str20; Where_To_Place : BYTE);

BEGIN
    CASE Where_To_Place OF
         Cndbase : BEGIN
                       Entity:= 'typing condition';
                       Fsize:= Filesize( Cbase)
                   END;
          Actbase : BEGIN
                       Entity:= 'typing action';
                       Fsize:= Filesize( Abase)
                   END;
    END
END;
{###################################################################}
```

```
{#####################################################################}
PROCEDURE Add_Text_Till_Done( Where_To_Place : BYTE);
{Used for addition of new condition text, this determines the}
{size of the condition base. (As starts at record 0, the file}
{size is always the same number as the position at  which the}
{new record would be appended.) Checks if user wants to add a}
{new condition text string, calling GET_AND_ENTER_TEXT if so.}
{Repeatedly does this until user finishes adding conditions. }
   VAR Pray_Continue : BOOLEAN;
               Entity : Str20;
                Fsize : INTEGER;

BEGIN
   REPEAT
      Menu_Head;
      Determine_Size_And_Entity( Fsize, Entity, Where_To_Place);
      WRITE('Ready to create text for ');
      WRITE(Entity,' #', (Fsize + 1));
      Pray_Continue:= Wants_To_Carry_On;
      IF Pray_Continue THEN
         Get_And_Enter_Text( Where_To_Place, Fsize)
   UNTIL (NOT Pray_Continue)
END;
{#####################################################################}



{#####################################################################}
PROCEDURE Sign_Off;

BEGIN
   Menu_Head;
   Lowvideo;
   WRITELN;
   WRITELN('You are now exitting high-level modifier module and being');
   WRITELN('returned to DOS. If you haven't yet written  up  routines');
   WRITELN('for  the  new   actions/conditions/indications  you  have ');
   WRITELN('declared the existence of, do so NOW.');
   WRITELN;
   WRITELN('If you are unhappy with one of your  additions here, just');
   WRITELN('delete the relevant file from the drive  you  are running');
   WRITELN('the program on, (normally RAM-drive), and restore the old');
   WRITELN('version.');
   WRITELN;
   WRITELN('Remember to save these files to floppy/hard-disk  if  you');
   WRITELN('are happy with what you've done.');
   Gotoxy(1,24);
   Normvideo;
   WRITE('Any key to continue.');
   Anykey:= Readkey;
END;
{#####################################################################}
```

```
{#####################################################################}
PROCEDURE Enter_Modifier;
{Determines whether user wishes to change old text or add}
{new text, and calls routines appropriate to the choice. }
  VAR Bid_Or_Play, Change_Or_New : CHAR;

BEGIN
   Basenam[1]:= 'BASEACTS.STA';
   Basenam[2]:= 'BASECNDS.STA';
   Ask_Type_Of_Modif( Change_Or_New);
   IF (Change_Or_New = 'C') THEN
        Change_Text
   ELSE BEGIN
        Stress_Operations_To_Perform;
        Add_Text_Till_Done( Cndbase);
        Add_Text_Till_Done( Actbase)
   END;
   Sign_Off
END;
{#####################################################################}




BEGIN              {Of HIGH_LEVEL_MODIFIER procedure.}
   Warn_Danger;
   IF (Knows_Way_In) THEN
      Enter_Modifier
END;
{#####################################################################}


END.
```

```
UNIT Ts6;
{This unit contains user-help routines.}

Interface

USES Ts1, Dos, Crt;

PROCEDURE Do_Help;


IMPLEMENTATION

{////////////////////////////////////////////////////////////////}
PROCEDURE Do_Help;
{This routine gives a menu from which the user may select the}
{particular area he/she  requires explanation  about.  On the}
{basis of that selection, the  appropriate  explanation  text}
{printing routine is called.                                  }
   VAR Valid_Help_Choices : Charset;
             Help_Choice : CHAR;



{////////////////////////////////////////////////////////////////}
PROCEDURE Show_Help_Options;

BEGIN
   Menu_Head;
   WRITELN('                        HELP  SECTION');
   Lowvideo;
   WRITELN;
   WRITELN('First I must find out in what area you have a problem. Below');
   WRITELN('is a MENU which shows the areas where help is available.');
   WRITELN('To get help in a particular area, simply press the letter to');
   WRITELN('the left of the '=' symbol (the highlighted letter).');
   WRITELN;
   WRITELN('Don't worry about mistakes - you can't hurt the machine.');
   WRITELN;
   WRITELN('Press the letter 'X' to exit this HELP section.');
   WRITELN;
   Menu_Write('I =  help on giving Instructions to system (teaching).');
   Menu_Write('L =  help on the Listing moule.');
   Menu_Write('N =  help on the Next-choice option.');
   Menu_Write('W =  help on the Why question option.');
   Menu_Write('X =  eXit help section.')
END;
{////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////}
PROCEDURE Get_Option_Carefully( VAR Opted : CHAR);
{Rather than the normal GET_VALID_CHOICE_FROM routine, this one}
{is nauseously user-friendly, on the grounds that if  the  user}
{needs help then probably not too expert in anything.         }

BEGIN
   Gotoxy(1,24);
   WRITE('Please press letter corresponding to one of options above : ');
   Opted:= Readkey;
   IF (NOT (Opted IN Valid_Help_Choices)) THEN
      REPEAT
         Gotoxy(1,24);
         Clreol;
         WRITE('You entered the letter ',Opted,' when you should ');
         WRITELN('have pressed I,L,N,W or X.');
         WRITE('Please try again : ');
         Opted:= Readkey;
      UNTIL (Opted IN Valid_Help_Choices)
END;
{////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Explain_Learner;

BEGIN
   WRITELN('      HELP  on  Instruction.');
   WRITELN;
   Lowvideo;
   WRITELN('The TrailStar system can be improved (or worsened!) by the');
   WRITELN('user in one of the following ways :');
   WRITELN;
   WRITELN('(1) You can add a new rule to check for certain  conditions,');
   WRITELN('    to use to determine a typing. (the action of the rule).');
   WRITELN('(2) You can delete an existing rule.');
   WRITELN('(3) You can add a condition to a rule.');
   WRITELN('(4) You can remove a condition from a rule.');
   WRITELN('(5) You can change the action of a rule (change what type is');
   WRITELN('    chosen if its conditions hold.');
   WRITELN('(6) You can change a rule's priority - the first rule  found');
   WRITELN('    where all conditions are true is the one used to  decide');
   WRITELN('    type. If you alter priority you alter the order in which');
   WRITELN('    the rules are looked at.');
   WRITELN;
   WRITELN('For a  full listing of all  conditions and actions currently');
   WRITELN('avilable on your system, use the Lister option from the main');
   WRITELN('menu - simply eXit from your current menu to get to main.');
   WRITELN;
   WRITELN('For more detail see  "The User's Guide to TrailStar".');
   Wait_Till_Read
END;
{////////////////////////////////////////////////////////////////////}


{////////////////////////////////////////////////////////////////////}
PROCEDURE Explain_Lister;

BEGIN
   WRITELN('      HELP  on  Lister option.');
   WRITELN;
   Lowvideo;
   WRITELN('The lister module allows the output of different lists to');
   WRITELN('screen or printer. The output sink can be toggled between');
   WRITELN('screen and printer by using the 'S' option in the  Lister');
   WRITELN('menu. The various listing possibilities are :');
   WRITELN('[1] Action listing - lists all actions  that  the  system');
   WRITELN('    caters for.');
   WRITELN('[2] Condition listing - lists all conditions catered for.');
   WRITELN('[3] Rule Listing - within this option you have three sub-');
   WRITELN('    options. You can get a listing of the conditions  and');
   WRITELN('    actions in a single rule with text describing  these,');
   WRITELN('    or a listing of all the rules  in  the  rule-base  in');
   WRITELN('    this  form,  or an  abbreviated listing of rules that');
   WRITELN('    just gives the numbers of conditions and actions  and');
   WRITELN('    not the text for them (a lot shorter than full list).');
   WRITELN('[4] Priority Listing - here a list of rule numbers in the');
   WRITELN('    descending order of their priority is output.');
   Wait_Till_Read
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Explain_Next_Choice;

BEGIN
   WRITELN('    HELP  on Next choice option.');
   Lowvideo;
   WRITELN;
   WRITELN('This option can only be used after  a  trail  typing  by the');
   WRITELN('computer to tell you what typing  the  computer  would  have');
   WRITELN('chosen if it hadn't made the typing it did. (So you can  see');
   WRITELN('what the next valid typing would have been, (people who know');
   WRITELN('a little about rule-based systems can read  that  as  "which');
   WRITELN('rule would be fired next) - this is handy when improving the');
   WRITELN('system by using the  Instruct  option  to teach the system a');
   WRITELN('better way to type.)');
   WRITELN;
   WRITELN('It is quite permissible to use the Next  choice  option  any');
   WRITELN('number of times in succession, thus getting the  3rd choice,');
   WRITELN('4th choice, etcetera. When the last choice was the last rule');
   WRITELN('whose conditions would hold true then you will get a message');
   WRITELN('to the effect that there was no next choice on using the 'N'');
   WRITELN('option.');
   Wait_Till_Read
END;
{////////////////////////////////////////////////////////////////////}


{////////////////////////////////////////////////////////////////////}
PROCEDURE Explain_Query;

BEGIN
   WRITELN('      HELP  on  Query  option.');
   Lowvideo;
   WRITELN;
   WRITELN('This option can only be used after a typing by the computer,');
   WRITELN('as its role is to give an  explanation of the  reasons  for');
   WRITELN('the typing made  -  this  constitutes  the  self-justifying');
   WRITELN('ability of TrailStar.');
   WRITELN;
   WRITELN('The way the system gives its explanations  is  by  showing');
   WRITELN('all the conditions that were in the rule fired on the last');
   WRITELN('play, along with the action the rule was obliged to follow');
   WRITELN('when these conditions were true. The rule number is  given');
   WRITELN('as an aid to the user attempting to Instruct the system.');
   WRITELN;
   Wait_Till_Read
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Give_Help_On( Option : CHAR);
{The branching routine which passes control to the}
{appropriate explanation printer.                  }
BEGIN
   IF (Option = 'X') THEN {Do nothing, exit wanted.}
   ELSE BEGIN
        Menu_Head;
        CASE Option OF
             'I' : Explain_Learner;
             'L' : Explain_Lister;
             'N' : Explain_Next_Choice;
             'W' : Explain_Query
        END
   END
END;
{////////////////////////////////////////////////////////////////////}
```

```
BEGIN                              {Of the DO_HELP overlay routine.}
    Valid_Help_Choices:= ['I','L','N','W','X'];
    REPEAT
        Show_Help_Options;
        Get_Option_Carefully( Help_Choice);
        Give_Help_On( Help_Choice)
    UNTIL (Help_Choice = 'X');
    Clrscr
END;
{////////////////////////////////////////////////////////////////////////}

END.
```

```
UNIT Ts7;
{This unit contains routines which implements the learning}
{function of the TrailStar system.                        }

Interface

USES Ts1, Dos, Crt;

PROCEDURE Learn;


IMPLEMENTATION


{-------------------------------------------------------------------}
PROCEDURE Learn;
{This routine allows the user/expert to add new rules, delete}
{old rules, add and delete conditions in  rules,  change rule}
{actions as well as altering the priority of rules.          }
TYPE Arr3_Of_Byte = ARRAY[1..3] OF BYTE;
            Str40 = String[40];

VAR  Learn_Num, Learn_B_Nam : Str12;
           Learn_Rule_Base : Rulefile;
            Learn_Pri_File : ARRAY[1..3] OF Int_File;
              Learn_Choice : CHAR;




{-------------------------------------------------------------------}
PROCEDURE Show_Learn_Options;

BEGIN
   Menu_Head;
   WRITELN('                        Learning Options');
   WRITELN;
   Lowvideo;
   Normvideo;
   WRITELN;
   Menu_Write('A =  Add a new rule.');
   Menu_Write('C =  Change action part of rule.');
   Menu_Write('D =  Delete an existing rule.');
   Menu_Write('E =  Eliminate condition from rule.');
   Menu_Write('I =  Insert new condition into rule.');
   Menu_Write('L =  List a rule.');
   Menu_Write('P =  change Priority of rule.');
   Menu_Write('X =  eXit learning module.');
   Gotoxy(1,25);
   WRITE('Please enter choice or 'H' for help : ');
   Learn_Choice:= Get_Valid_Choice_From( ['A','C','D','E','I','L','P','X']);
END;
{-------------------------------------------------------------------}
```

```
{////////////////////////////////////////////////////////////////////////}
PROCEDURE Give_Learn_Help;

BEGIN
   WRITELN('      HELP  on  Instruction.');
   WRITELN;
   Lowvideo;
   WRITELN('The TrailStar system can be improved (or worsened!) by the');
   WRITELN('user in one of the following ways :');
   WRITELN;
   WRITELN('(1) You can add a new rule to check for certain  conditions,');
   WRITELN('    to use to determine a typing. (the action of the rule).');
   WRITELN('(2) You can delete an existing rule.');
   WRITELN('(3) You can add a condition to a rule.');
   WRITELN('(4) You can remove a condition from a rule.');
   WRITELN('(5) You can change the action of a rule (change what type is');
   WRITELN('    chosen if its conditions hold.');
   WRITELN('(6) You can change a rule's priority - the first rule  found');
   WRITELN('    where all conditions are true is the one used to  decide');
   WRITELN('    type. If you alter priority you alter the order in which');
   WRITELN('    the rules are looked at.');
   WRITELN;
   WRITELN('For a  full listing of all  conditions and actions currently');
   WRITELN('avilable on your system, use the Lister option from the main');
   WRITELN('menu - simply eXit from your current menu to get to main.');
   WRITELN;
   WRITELN('For more detail see  "The User's Guide to TrailStar".');
   Wait_Till_Read
END;
{////////////////////////////////////////////////////////////////////////}



{----------------------------------------------------------------------}
PROCEDURE Get_Legal_Integer( VAR Intval : INTEGER; Colst, Rowst : BYTE);
{Gets an integer from user in string form (so can test whether it is}
{syntactically okay without crashing with a  run-time  error), tests}
{and repeats prompt and retrieval if user did not enter a valid int.}
    VAR Errflag : INTEGER;
          Instr : Str12;

BEGIN
   REPEAT
        Gotoxy(Colst,Rowst);
        Clreol;
        READLN(Instr);
        VAL(Instr,Intval,Errflag);
        IF (Errflag <> 0) THEN BEGIN
            Gotoxy(1,Rowst);
            WRITE('Invalid integer - retry : ');
            Clreol
        END
   UNTIL (Errflag = 0)
END;
{----------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Get_Valid_Rule_Num( VAR R_Num : INTEGER; VAR Rule_Got : Rules);
{Gets rule number from user via GET_LEGAL_INTEGER, and then checks to}
{see if rule exists. If not then sets R_NUM to 0, else  R_NUM  set to}
{rule number chosen and RULE_GOT to rule this corresponds to.        }

BEGIN
   Clear_From( 24);
   WRITE('Please enter rule number : ');
   Get_Legal_Integer(R_Num,28,24);

   {$I-}
   SEEK( Rbase, (R_Num - 1));
   IF (Ioresult <> 0) THEN R_Num:= 0
   ELSE BEGIN
        READ( Rbase, Rule_Got);
        IF (Ioresult <> 0) THEN R_Num:= 0
   END;
   {$I+}

   Gotoxy(1,24);
   Clreol
END;
{-------------------------------------------------------------------}




{-------------------------------------------------------------------}
PROCEDURE Get_Valid_Cond_Num( VAR Cnum : INTEGER; VAR C_Str : Str80);
{Like GET_VALID_RULE_NUM except here it is the condition base which}
{is used rather than the rule-base, and so a text string instead of}
{a rule is retrieved.                                               }

BEGIN
   Clear_From( 24);
   WRITE('Please enter condition number : ');
   Get_Legal_Integer( Cnum,33,24);

   {$I-}
   SEEK( Cbase, (Cnum - 1));
   IF (Ioresult <> 0) THEN Cnum:= 0
   ELSE BEGIN
        READ( Cbase, C_Str);
        IF (Ioresult <> 0) THEN Cnum:= 0
   END;
   {$I+}

   Gotoxy(1,24);
   Clreol
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Get_Valid_Act_Num( VAR Actnum : INTEGER; VAR Act_Str : Str80);
{Like GET_VALID_RULE_NUM except here it is the action base which}
{is used rather than the rule-base, and so a text string instead}
{of a rule is retrieved.                                         }
BEGIN
   Clear_From( 24);
   WRITE('Please enter action number : ');
   Get_Legal_Integer( Actnum,30,24);

   {$I-}
   SEEK( Abase, (Actnum - 1));
   IF (Ioresult <> 0) THEN Actnum:= 0
   ELSE BEGIN
        READ( Abase, Act_Str);
        IF (Ioresult <> 0) THEN Actnum:= 0
   END;
   {$I+}

   Gotoxy(1,24);
   Clreol
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Show_The_Rule( Rule_In : Rules);
{Calls routines to write all the conditions and the action in}
{the rule passed as argument.                                }
   VAR I : INTEGER;

BEGIN
   WRITELN('CONDITIONS');
   FOR I:= 1 TO Conds_In( Rule_In) DO BEGIN
       WRITE('[',I:2,']');
       Show_Cond( Rule_In[I])
   END;
   WRITE('ACTION : ');
   Show_Act( Rule_In[20])
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Opt_Write( Num : BYTE; St_1 : Str40; Ch : CHAR; St_2 : Str40);
{Used to print the parameter option menu, this ensures that the second}
{two parameters appear on the right of the screen, first two on  left,}
{and that first and third parameters are highlighted.                  }
     VAR I : BYTE;

BEGIN
     Normvideo;
     WRITE(Num);
     Lowvideo;
     WRITE('  =  ');
     WRITE(St_1);
     FOR I:= (6 + LENGTH(St_1)) TO 40 DO
         WRITE(' ');
     Normvideo;
     WRITE(Ch);
     Lowvideo;
     WRITE('  =  ');
     WRITELN(St_2)
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Find_Which_File( VAR Rnum_In, Pos_In_File : INTEGER);
{Finds which priority file the rule number RNUM_IN is kept in,}
{and returns the number of this file, (priority files an array}
{of files), as well as the position the rule number found at. }
     VAR Fsize, Rnum_At_Pos : INTEGER;
                 Found : BOOLEAN;

BEGIN
   Found:= FALSE;
   RESET( Pribase);
   Fsize:= Filesize( Pribase);
   Pos_In_File:= -1;
   WHILE ((Pos_In_File < Fsize) AND (NOT Found)) DO BEGIN
         Pos_In_File:= Pos_In_File + 1;
         READ( Pribase, Rnum_At_Pos);
         IF (Rnum_At_Pos = Rnum_In) THEN
             Found:= TRUE
   END
END;
{-------------------------------------------------------------------}




{-------------------------------------------------------------------}
PROCEDURE Get_Rule2_And_Position_And_Check_Valid
  ( VAR Alt_Choice : CHAR; VAR Pos2 : INTEGER);
{Gets user's choice of a second rule, and reads entries in the}
{priority file in which the first rule was found until  either}
{the second rule number is found or the end of  priority  file}
{is reached. In latter case error message sent.                }
         VAR R2_Num, Cnt, Fsize, Num_Read : INTEGER;

BEGIN
   Gotoxy(1,24);
   Clreol;
   WRITE('Please enter number of second rule : ');
   Get_Legal_Integer( R2_Num, 38, 24);
   Gotoxy(1,24);
   Clreol;
   Pos2:= -1;
   Cnt:= -1;
   RESET( Pribase);
   Fsize:= Filesize( Pribase);
   WHILE ((Pos2 = -1) AND (Cnt <= Fsize)) DO BEGIN
         Cnt:= Cnt + 1;
         READ( Pribase, Num_Read);
         IF (Num_Read = R2_Num) THEN Pos2:= Cnt
   END;

   IF (Pos2 = -1) THEN BEGIN
       Gotoxy(1,24);
       WRITE('Invalid - 2nd rule not in priority file.');
       WRITE('Any key to continue.');
       Anykey:= Readkey;
       Alt_Choice:= 'X'        {Abort signal to caller.}
   END
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Swop_Pri (Pos1, Pos2 : INTEGER);
{Swops the rule numbers held at POS1 and POS2 in priority file PFIL.}
   VAR Val1, Val2 : INTEGER;

BEGIN
   SEEK( Pribase, Pos1);
   READ( Pribase, Val1);
   SEEK( Pribase, Pos2);
   READ( Pribase, Val2);
   SEEK( Pribase, Pos1);
   WRITE(Pribase, Val2);
   SEEK( Pribase, Pos2);
   WRITE(Pribase, Val1)
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Set_Above( Pos1, Pos2 : INTEGER);
{Sets rule number at entry POS1 into position above POS2 - above}
{meaning of higher priority  =>  earlier in file -  altering the}
{positions of numbers between them down.                         }
   VAR I : INTEGER;

BEGIN
   IF (Pos1 <= Pos2) THEN
        {Do nothing, POS1 entry already has higher priority.}
   ELSE
        FOR I:= Pos1 DOWNTO (Pos2 + 1) DO
            Swop_Pri( I, (I-1))
            {'Bubbles' val at pos1 up to pos2.}
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Set_Below( Pos1, Pos2 : INTEGER);
{Inverse of SET_ABOVE described above!}
   VAR I : INTEGER;

BEGIN
   IF (Pos1 >= Pos2) THEN
        {Do nothing, POS1 entry already has lower priority.}
   ELSE
        FOR I:= Pos1 TO (Pos2 - 1) DO
            Swop_Pri( I, (I+1))
            {'Bubbles' val at pos1 down to pos2.}
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Handle_Extreme_Priority( Pos1 : INTEGER; Hilo : BOOLEAN);
{Sets POS2 to 1 and calls SET_ABOVE to ensure POS1 value set in entry}
{above this if highest priority wanted,  else to number of  the  last}
{record in file and use SET_BELOW if lowest priority wanted.         }
    VAR Pos2 : INTEGER;

BEGIN
   IF (Hilo = High) THEN BEGIN
       Pos2:= 0;
       Set_Above( Pos1, Pos2)
   END
   ELSE BEGIN
       Pos2:= Filesize( Pribase) - 1;
       Set_Below( Pos1, Pos2)
   END;

   Gotoxy(1,25);
   WRITE('Extreme priority setting made - any key to continue.');
   Anykey:= Readkey;
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Handle_Two_Rules( Alt_Choice : CHAR; Pos1 : INTEGER);
{Handles cases where user wants to set a rule's  priority relevant}
{in some way to the priority of another rule.  First  gets  second}
{rule number. If this valid then passes control to the appropriate}
{routine, and sends message telling user operation done okay.     }
   VAR  Pos2 : INTEGER;

BEGIN
   Get_Rule2_And_Position_And_Check_Valid( Alt_Choice, Pos2);

   CASE Alt_Choice OF
        'A' : Set_Above( Pos1, Pos2);
        'B' : Set_Below( Pos1, Pos2);
        'S' : Swop_Pri( Pos1, Pos2);
        'X' : {Do nothing - abort called for.}
   END;

   IF (Alt_Choice <> 'X') THEN BEGIN
       Gotoxy(1,25);
       Clreol;
       WRITE('Operation successfully completed.  Press any key.');
       Anykey:= Readkey;
   END
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Alter_To_Choice( R_Num, Pos1 : INTEGER);
{Used for priority alteration, this shows user the various ways}
{in which rule priority can be changed, and gets user's choice.}
{Then passes control to routine appropriate to that choice.    }
    VAR Alt_Choice : CHAR;

BEGIN
   Menu_Write('A =  set priority Above that of another rule.');
   Menu_Write('B =  set priority Below that of another rule.');
   Menu_Write('L =  give rule Lowest possible priority.');
   Menu_Write('H =  give rule Highest possible priority.');
   Menu_Write('S =  Swop rule's priority with that of another rule.');

   Gotoxy(1,20);
   WRITE('Please enter choice : ');
   Alt_Choice:= Get_Valid_Choice_From(['A','B','L','H','S']);

   IF (Alt_Choice IN ['A','B','S']) THEN
        Handle_Two_Rules( Alt_Choice, Pos1)
   ELSE
       IF (Alt_Choice = 'L') THEN
            Handle_Extreme_Priority( Pos1, Low)
       ELSE Handle_Extreme_Priority( Pos1, High)
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Alter_Rule_Priority;
{Gets the number of the rule whose priority is to be altered,}
{determines which priority file the rule is  in,  then  calls}
{ALTER_TO_CHOICE to find type of  alteration  wanted  and  to}
{implement alteration. Sees if user then wants to  alter  the}
{priority of another rule, and repeats if so.                 }
   VAR Pfil, R_Num, Pos_In_File : INTEGER;
                    Pr_Choice : CHAR;
                        R_Got : Rules;

BEGIN
   REPEAT
       Menu_Head;
       WRITELN('     PRIORITY HANDLER.');
       Lowvideo;
       Get_Valid_Rule_Num( R_Num, R_Got);
       Find_Which_File( R_Num, Pos_In_File);
       Gotoxy(1, 5);
       Alter_To_Choice( R_Num, Pos_In_File);
       Gotoxy(1,25);
       Clreol;
       WRITE('Change another priority (Y/N) ? ');
       Pr_Choice:= Get_Valid_Choice_From( Yesno)
  UNTIL (Pr_Choice = 'N')
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Init_Kill_Screen( R_Num : INTEGER; Rule_Got : Rules);
{Prints screen heading for condition deletion routine, and calls}
{SHOW_THE_RULE so user can see rule involved.                      }

BEGIN
   Menu_Head;
   WRITELN('            Condition deletion in rule ',R_Num);
   Lowvideo;
   Show_The_Rule( Rule_Got);
   Gotoxy(1,24)
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Say_Cant_Kill_Last;

BEGIN
   WRITELN('Can't delete last condition from rule without deleting');
   WRITE(' rule itself - any key to continue.');
   Anykey:= Readkey;
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Get_Valid_Cond_To_Kill( VAR Dead_Cond : INTEGER;
                                      Max_Cond : INTEGER);
{Repeatedly prompts user for integer until get one within}
{the bounds of possible conditions to delete. (Note: here}
{the user sees the rule with conditions numbered in terms}
{of their ordinal position in the current  rule,  so  are}
{getting  this  ordinal  position,  NOT the number of the}
{condition in the condition base.)                        }

BEGIN
   REPEAT
       Gotoxy(1,24);
       Clreol;
       WRITE('Please enter number of condition to delete : ');
       Get_Legal_Integer( Dead_Cond, 46, 24)
   UNTIL ((Dead_Cond > 0) AND (Dead_Cond <= Max_Cond));

   Gotoxy(1,24);
   Clreol
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Kill_Cond_In_Rule;
{First get the  rule number user wants to delete a condition from.}
{If only  one condition in rule then tell user that cannot  delete}
{this (he/she free to delete rule, but not to have 'unconditional'}
{rule.  Show  the  rule  to user, and find which condition number,}
{(number in terms of ordinal position  of condition in rule), user}
{wants deleted. Delete cond by 'sliding all conds above it down 1'}
{and then write this new version of the rule out to the rule base.}
   VAR R_Num, Dead_Cond, I : INTEGER;
            Kill_Choice : CHAR;
               Rule_Got : Rules;

BEGIN
   REPEAT
      Get_Valid_Rule_Num( R_Num, Rule_Got)
   UNTIL (R_Num <> 0);

   REPEAT
      Init_Kill_Screen( R_Num, Rule_Got);
      IF (Conds_In( Rule_Got) = 1) THEN BEGIN
          Say_Cant_Kill_Last;
          Kill_Choice:= 'N'
      END
      ELSE BEGIN
          Get_Valid_Cond_To_Kill( Dead_Cond, Conds_In( Rule_Got));
          IF (Dead_Cond <> Conds_In( Rule_Got)) THEN
          {If not last rule being killed must shift those above down.}
              FOR I:= Dead_Cond TO (Conds_In( Rule_Got) - 1) DO
                  Rule_Got[I]:= Rule_Got[I+1];
          {Now 'delete' last rule by altering rules cond count.      }
          Rule_Got[0].Cond_Num:= Conds_In( Rule_Got) - 1;

          {Finally write altered rule to file.                       }
          SEEK( Rbase, (R_Num - 1));
          WRITE( Rbase, Rule_Got);

          Gotoxy(1,24);   Clreol;
          WRITE('Kill another condition in this rule (Y/N) ? ');
          Kill_Choice:= Get_Valid_Choice_From( Yesno)
      END
   UNTIL (Kill_Choice = 'N')
END;
{-------------------------------------------------------------------}




{-------------------------------------------------------------------}
PROCEDURE Show_Parm_Choices;
{Gives a menu showing the various parameter choices feasible}
{as parameters of conditions/actions.                       }

BEGIN
   Menu_Head;
   WRITELN('                    PARAMETER CHOICES.');
   WRITELN;
   Menu_Write('I =  Cardinal number.');
   Menu_Write('J =  Ordinal number (xth Lowest/Highest).');
END;
{-------------------------------------------------------------------}
```

```
FUNCTION Get_Number_In_Range( St_Range, End_Range : BYTE) : INTEGER;
    VAR Err, Num_In : INTEGER;
            Num_Str : Str12;

BEGIN
  REPEAT
    Gotoxy(1,14);
    Clreol;
    Gotoxy(1,14);
    WRITE('Please enter number in range ',St_Range,'->',End_Range,' : ');
    READLN( Num_Str);
    VAL( Num_Str, Num_In, Err);
    IF (Err = 0) THEN
       IF ((Num_In < St_Range) OR (Num_In > End_Range)) THEN
           Err:= 1
  UNTIL (Err = 0);
  Get_Number_In_Range:= Num_In
END;



{-------------------------------------------------------------------}
FUNCTION Get_Parm_Choice (Of_Type : BYTE) : BYTE;
{Calls routines to display parameter choices and get user's}
{choice. IF choice fully defined then returns a value which}
{is understood by system to  show  the  particular  choice,}
{else gets particular  'choice within choice'  wanted. (For}
{example, if original choice was 'Card' then  need  to  get}
{the actual card involved before can determine parameter.  }
   VAR Local_Char, Parm_Ch : CHAR;
         Intval : INTEGER;


BEGIN
   Show_Parm_Choices;
   Gotoxy(1,10);
   WRITE('Please enter choice of parameter type : ');
   Local_Char:= Get_Valid_Choice_From(['I','J']);
   WRITE( Local_Char);
   IF (Local_Char = 'I') THEN
        Intval:= Get_Number_In_Range( 0,100)
   ELSE Intval:= Get_Number_In_Range( 1,100);
   IF (Local_Char = 'I') THEN
        Get_Parm_Choice:= Intval
   ELSE Get_Parm_Choice:= Intval + 100
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Set_Upcase( VAR St_In : Str80);
{Sets the string ST_IN to its uppercase equivalent.}
   VAR I : BYTE;

BEGIN
   FOR I:= 1 TO LENGTH( St_In) DO
       St_In[I]:= Upcase( St_In[I])
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Get_Text_Parms( Txt_Str : Str80; VAR Parm_Cnt : BYTE;
                         VAR Parm_Types : Arr3_Of_Byte);
{Finds all parameter flags in the text string (from condition or}
{action base) passed as parameter, and sets the PARM_TYPES array}
{to values showing the particular flag type involved.           }
    VAR Possible_Parm : Str80;
                  Pos : BYTE;

BEGIN
   Parm_Cnt:= 0;
   Pos:= 1;
   WHILE (NOT (Eo_St( Txt_Str, Pos))) DO BEGIN
      Get_Item( Possible_Parm, Pos, Txt_Str);
      IF (Possible_Parm[1] = '!') THEN BEGIN
         Parm_Cnt:= Parm_Cnt + 1;
         Set_Upcase( Possible_Parm);
      END
   END;
END;
{-------------------------------------------------------------------}




{-------------------------------------------------------------------}
PROCEDURE Init_Cond_Add(VAR Rule_In : Rules; VAR Cnd_Str : Str80;
                                    VAR C_Num, New_Ent : INTEGER);
{Gets the number (condition base number) of the condition to be}
{added to rule, increments the counter of number of  conditions}
{in the rule, and places the number of the new condition in the}
{rule at position defined by new counter value.                 }


BEGIN
   REPEAT
      Get_Valid_Cond_Num( C_Num, Cnd_Str)
   UNTIL (C_Num <> 0);

   New_Ent:= Conds_In(Rule_In) + 1;
   Rule_In[0].Cond_Num:= New_Ent;
   Rule_In[ New_Ent].Cond_Num:= C_Num;
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Add_Cond_To_Rule( VAR Rule_In : Rules; VAR Abort : BOOLEAN);
{Finds what condition to add to rule, and gets all the parameters}
{needed by condition before storing new condition.                 }
    VAR Parm_Cnt, Parms_Got : BYTE;
                Add_C_Choice : CHAR;
                C_Pos, C_Num : INTEGER;
                  Parm_Types : Arr3_Of_Byte;
                     Cnd_Str : Str80;


BEGIN
   Init_Cond_Add( Rule_In, Cnd_Str, C_Num, C_Pos);
   Get_Text_Parms( Cnd_Str, Parm_Cnt, Parm_Types);
   Parms_Got:= 0;
   REPEAT
       Menu_Head;
       WRITELN('New condition to be added is : ');
       Lowvideo;
       WRITELN(Cnd_Str);
       WRITELN;
       IF (Parm_Cnt <> Parms_Got) THEN BEGIN
           WRITELN('"!" symbols indicate parameters.');
           WRITE('You will now be prompted for parameter ');
           WRITELN(Parms_Got + 1)
       END;
       Gotoxy(1,24);
       WRITE('Enter 'A' to abort, any other key to continue.');
       Add_C_Choice:= Upcase( Readkey);
       IF ((Add_C_Choice <> 'A') AND (Parms_Got < Parm_Cnt)) THEN BEGIN
           Parms_Got:= Parms_Got + 1;
           Rule_In[ C_Pos].C_Parms[ Parms_Got]:=
                   Get_Parm_Choice( Parm_Types[ Parms_Got])
       END
   UNTIL ((Parm_Cnt = Parms_Got) OR (Add_C_Choice = 'A'));

   IF (Add_C_Choice = 'A') THEN
       Abort:= TRUE
   ELSE Abort:= FALSE
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Insert_Condition_In_Rule;
{Gets rule number user wants to add condition to, displays the}
{rule, and checks that condition addition would not  result in}
{attempt to have more than 19 conditions in rule.  If  such an}
{attempt is being made then sends error message, else calls on}
{ADD_COND_TO_RULE routine to get and add the  condition.  Then}
{writed the new version of the  rule  to  the  rule  base, and}
{checks to see if user wants to add another condition.        }
   VAR Ins_Choice : CHAR;
       Rule_Got : Rules;
        Aborted : BOOLEAN;
          R_Num : INTEGER;

BEGIN
   REPEAT
      Get_Valid_Rule_Num( R_Num, Rule_Got)
   UNTIL (R_Num <> 0);

   REPEAT
      Menu_Head;
      WRITELN('              Condition addition to rule ',R_Num);
      Lowvideo;
      Show_The_Rule( Rule_Got);
      IF (Conds_In( Rule_Got) = 19) THEN BEGIN
         Gotoxy(1,24);
         Clreol;
         WRITELN('Can't have more than 19 conditions in a rule.');
         WRITE('Aborted - press any key to continue.');
         Anykey:= Readkey;
         Ins_Choice:= 'N'
      END
      ELSE BEGIN
         Add_Cond_To_Rule( Rule_Got, Aborted);
         IF (NOT Aborted) THEN BEGIN
              SEEK( Rbase, (R_Num - 1));
              WRITE( Rbase, Rule_Got)
         END
         ELSE Rule_Got[0].Cond_Num:= Conds_In( Rule_Got) - 1;
         Gotoxy(1,24);
         Clreol;
         WRITE('Add another condition (Y/N) ? ');
         Ins_Choice:= Get_Valid_Choice_From( Yesno)
      END
   UNTIL (Ins_Choice = 'N')
END;
{-------------------------------------------------------------------}




{-------------------------------------------------------------------}
PROCEDURE Init_Act_Change( VAR Rule_In : Rules; VAR Act_Str : Str80;
                           VAR Act_Num : INTEGER);
{Gets the number of the new action user desires for rule, and}
{sets the part of the rule defining the action accordingly.  }

BEGIN
   REPEAT
      Get_Valid_Act_Num( Act_Num, Act_Str)
   UNTIL (Act_Num <> 0);
   Rule_In[20].Cond_Num:= Act_Num;
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Get_New_Action_For( VAR Rule_In : Rules; VAR Abort : BOOLEAN);
{After finding the number of the new action for the rule, scans the}
{action base text to pick up parameter flags and gets  values  user}
{wants for parameters, storing results in action part of rule.     }
    VAR Parm_Cnt, Parms_Got : BYTE;
                Parm_Types : Arr3_Of_Byte;
                Act_Choice : CHAR;
                   Act_Num : INTEGER;
                   Act_Str : Str80;

BEGIN
   Init_Act_Change( Rule_In, Act_Str, Act_Num);
   Get_Text_Parms( Act_Str, Parm_Cnt, Parm_Types);
   Parms_Got:= 0;
   REPEAT
       Menu_Head;
       WRITE('ACTION : ');
       Lowvideo;
       WRITELN(Act_Str);
       WRITELN;
       IF (Parm_Cnt <> Parms_Got) THEN BEGIN
           WRITELN('"!" symbols indicate parameters.');
           WRITE('You will now be prompted for parameter ');
           WRITELN(Parms_Got + 1)
       END;
       Gotoxy(1,24);
       WRITE('Enter 'A' to abort, any other key to continue.');
       Act_Choice:= Upcase( Readkey);
       IF ((Act_Choice <> 'A') AND (Parms_Got < Parm_Cnt)) THEN BEGIN
           Parms_Got:= Parms_Got + 1;
           Rule_In[20].C_Parms[ Parms_Got]:=
                Get_Parm_Choice( Parm_Types[ Parms_Got])
       END
   UNTIL ((Parm_Cnt = Parms_Got) OR (Act_Choice = 'A'));

   IF (Act_Choice = 'A') THEN
        Abort:= TRUE
   ELSE Abort:= FALSE
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Change_Rule_Action;
{Calls routines to get rule number action change desired in,}
{and then add action, along with user chosen  parameters, to}
{the rule. Then stores new version of rule in rule base.    }
     VAR Rule_Got : Rules;
          Aborted : BOOLEAN;
          Confirm : CHAR;
            R_Num : INTEGER;

BEGIN
   REPEAT
       Get_Valid_Rule_Num( R_Num, Rule_Got)
   UNTIL (R_Num <> 0);

   Menu_Head;
   WRITELN('              ACTION CHANGE FOR RULE ',R_Num);
   Lowvideo;
   Show_The_Rule( Rule_Got);
   Gotoxy(1,25);
   WRITE('This the rule to change (Y/N) ? ');
   Confirm:= Get_Valid_Choice_From( Yesno);
   IF (Confirm = 'Y') THEN BEGIN
      Get_New_Action_For( Rule_Got, Aborted);
      IF (NOT Aborted) THEN BEGIN
         SEEK( Rbase, (R_Num - 1));
         WRITE( Rbase, Rule_Got)
      END
   END
ND;
-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Show_And_Check_Sure( R_Num : INTEGER; Rule_In : Rules;
                               VAR Del_Choice : CHAR);
{Shows the rule whose number the user has entered in response to a}
{prompt as to number of rule to delete, and checks that  user sure}
{that this is the rule he/she wished to delete, setting DEL_CHOICE}
{accordingly as signal to caller.                                 }

BEGIN
   Gotoxy(1,3);
   Clreol;
   WRITELN('           Rule ',R_Num,' to be deleted.');
   Lowvideo;
   Show_The_Rule( Rule_In);
   Gotoxy(1,25);
   Clreol;
   WRITE('Sure you want to delete this rule (Y/N) ? ');
   Del_Choice:= Get_Valid_Choice_From( Yesno)
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Compact_Rule_Base( R_Num : INTEGER);
{On deletion of rule have to physically remove it from the}
{rule base file. Do this by copying entire rule  base to a}
{temporary file, scrubbing, (rewriting), the rulebase, and}
{then copying all the rules bar that deleted back from the}
{temporary file.                                          }
   VAR Temp_Base : FILE OF Rules;
       Rule_Rec : Rules;
              I : INTEGER;

BEGIN
   ASSIGN(Temp_Base,'TEMPBASE');
   REWRITE(Temp_Base);
   RESET( Rbase);

   WHILE (NOT (EOF (Rbase))) DO BEGIN
         READ( Rbase, Rule_Rec);
         WRITE(Temp_Base,Rule_Rec)
   END;

   CLOSE( Rbase);
   CLOSE( Temp_Base);

   ASSIGN( Rbase, 'RULES' + Curbase + '.STA');
   REWRITE( Rbase);
   ASSIGN( Temp_Base, 'TEMPBASE');
   RESET( Temp_Base);

   {Copy all rules bar deleted one back.}
   FOR I:= 0 TO (R_Num - 2) DO BEGIN
       READ( Temp_Base, Rule_Rec);
       WRITE( Rbase, Rule_Rec)
   END;
   READ( Temp_Base, Rule_Rec);          {Skipping over 'dead' record.}
   WHILE (NOT (EOF (Temp_Base))) DO BEGIN
       READ( Temp_Base, Rule_Rec);
       WRITE( Rbase, Rule_Rec)
 END;

ND;
-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Compact_Pri_File( Where_Killed : INTEGER);
{Compaction of priority file to delete the rule number corresponding}
{to a rule deleted is done by copying all entries bar that  which is}
{the number of the deleted rule to a temporary file,  rewriting  the}
{priority file, and then copy the temporary file back.              }
   VAR R_Num, Fsize, I : INTEGER;
                Tempfil : FILE OF INTEGER;


BEGIN
   ASSIGN( Tempfil,'TEMPPRI');
   REWRITE( Tempfil);
   RESET( Pribase);
   FOR I:= 0 TO (Where_Killed - 1) DO BEGIN
       READ( Pribase, R_Num);
       WRITE( Tempfil, R_Num)
   END;
   READ( Pribase, R_Num);   {Skips over 'dead' entry.}
   WHILE (NOT (EOF ( Pribase))) DO BEGIN
       READ( Pribase, R_Num);
       WRITE( Tempfil, R_Num)
   END;
   CLOSE( Tempfil);
   CLOSE( Pribase);
   ASSIGN( Tempfil, 'TEMPPRI');
   RESET( Tempfil);
   ASSIGN( Pribase, 'PRIOR' + Curbase + '.STA');
   REWRITE( Pribase);
   {Now copy back file without 'dead' entry.}
   Fsize:= Filesize( Tempfil);
   FOR I:= 1 TO Fsize DO BEGIN
       READ( Tempfil, R_Num);
       WRITE( Pribase, R_Num)
   END;

END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Kill_Priority_Entry( R_Num : INTEGER);
{Goes through all priority file entries. Any rule number found}
{which is greater than the number of the rule to be deleted is}
{going to have a number one lower on deletion, and so  must be}
{altered in the priority file entry. Also the file  and  place}
{at which the 'dead' rule is found are marked and sent  to the}
{COMPACT_PRI_FILE routine which handles the elimination of the}
{rule number from the priority file in which it was in.       }
    VAR Fil, Place, Pri_Found, How_Many : INTEGER;
                Death_Place, Death_File : INTEGER;

BEGIN
   RESET( Pribase);
   How_Many:= Filesize( Pribase);
   Place:= 0;
   WHILE (NOT (EOF( Pribase))) DO BEGIN
        READ( Pribase, Pri_Found);
        IF (Pri_Found < R_Num) THEN
                {Do nothing - this rule number won't be affected.}
        ELSE
          IF (Pri_Found > R_Num) THEN BEGIN
             {Compacted rule base => num 1 less than previously.}
             Pri_Found:= Pri_Found - 1;
             SEEK( Pribase, Place);
             WRITE( Pribase,Pri_Found)
          END
          ELSE Death_Place:= Place;  {Rule being deleted found.}
        Place:= Place + 1;
   END;
   Compact_Pri_File( Death_Place)
ND;
-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Kill_Rule_File_Entries( R_Num : INTEGER);
{Driver for rule deletion, calls routines to delete rule from}
{rule base and delete rule number from priority file.        }

BEGIN
   Gotoxy(1,24);
   Clreol;
   WRITE('Deleting rule......');
   Compact_Rule_Base( R_Num);
   Kill_Priority_Entry( R_Num);
   Gotoxy(1,24);
   Clreol;
   WRITE('Deletion done.')
END;
{-------------------------------------------------------------------}




{-------------------------------------------------------------------}
PROCEDURE Delete_Rule;
{Repeatedly calls on user for number of rule to delete, shows}
{user rule, and checks that rule is in fact one user wants to}
{delete. If it is then calls routine to handle actual delete.}
{Then prompts user to see if wants to  delete  another  rule,}
{continuing until user responds negatively to prompt.        }
   VAR Del_Choice : CHAR;
       Rule_Back : Rules;
           R_Num : INTEGER;

BEGIN
   REPEAT
      Menu_Head;
      WRITELN('                    RULE DELETION');
      Lowvideo;
      Get_Valid_Rule_Num( R_Num, Rule_Back);

      IF (R_Num <> 0) THEN BEGIN
          Show_And_Check_Sure( R_Num, Rule_Back, Del_Choice);
          IF (Del_Choice = 'Y') THEN  {If user confirmed delete.}
              Kill_Rule_File_Entries( R_Num)
      END
      ELSE WRITELN('No such rule!!!');

      Gotoxy(1,25);
      Clreol;
      WRITE('Delete another (Y/N) ? ');
      Del_Choice:= Get_Valid_Choice_From( Yesno)
   UNTIL (Del_Choice = 'N')
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Creation_Head( Rule_In : Rules);
{Header for rule creation, writes conditions so far in rule.}

BEGIN
   Menu_Head;
   WRITELN('                      RULE  CREATION');
   Lowvideo;
   IF (Conds_In( Rule_In) <> 0) THEN BEGIN
       WRITELN('CONDITIONS :');
       Write_Conds( Rule_In)
   END
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
FUNCTION More_Conds_Wanted : CHAR;
{Checks if user wishes to add more conditions or not or to abort.}
   VAR Local_Char : CHAR;

BEGIN
   Gotoxy(1,25);
   WRITE('More conditions (Y/N or A to abort creation) ? ');
   Local_Char:= Get_Valid_Choice_From(['Y','N','A']);
   More_Conds_Wanted:= Local_Char
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Get_All_Conds(VAR Rule_Making : Rules; VAR Add_Choice : CHAR);
{Repeatedly calls ADD_COND_TO_RULE until either user indicates that}
{no more conditions wanted or 19 conditions entered for rule.      }
    VAR Cnd_Cnt : BYTE;
        Abort : BOOLEAN;

BEGIN
   Cnd_Cnt:= 0;
   REPEAT
       Add_Cond_To_Rule( Rule_Making, Abort);
       IF (NOT Abort) THEN Cnd_Cnt:= Cnd_Cnt + 1;
       Creation_Head( Rule_Making);
       IF (Cnd_Cnt <> 19) THEN
           Add_Choice:= More_Conds_Wanted
   UNTIL ((Add_Choice IN ['N','A']) OR (Cnd_Cnt = 19))
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Show_Rule_And_Check_Ok( Rule_In : Rules; VAR Abort : BOOLEAN);
{Shows what the rule ready to be stored looks like, and gets user}
{to say whether or not system should go ahead and store rule.     }
   VAR Decision : CHAR;

BEGIN
   Menu_Head;
   WRITELN('       RULE CREATION');
   Lowvideo;
   Show_The_Rule( Rule_In);
   Gotoxy(1,25);
   WRITE('Rule ready to store. "A" to abort, else any other key.');
   Decision:= Upcase( Readkey);
   IF (Decision = 'A') THEN
        Abort:= TRUE
   ELSE Abort:= FALSE
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Store_In_Rule_Base( Rule_Made : Rules; VAR R_Num : INTEGER);
{Writes new rule to rule base at end of rule base.}

BEGIN
   R_Num:= Filesize( Rbase);
   SEEK( Rbase, R_Num);
   WRITE( Rbase, Rule_Made)
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Find_And_Set_Rule_Priority( R_Num : INTEGER);
{Finds priority file needed for rule, and writes rule number}
{to end of file. Then calls ALTER_TO_CHOICE routine  to  set}
{the priority entry to position user desires.                }
   VAR Pfil, Fsize : INTEGER;

BEGIN
   Fsize:= Filesize( Pribase);
   SEEK( Pribase, Fsize);
   R_Num:= R_Num + 1;
   WRITE( Pribase, R_Num );       {Insert at EOF.       }
   Menu_Head;
   WRITELN('   PRIORITY  CHOICES.');
   Lowvideo;
   WRITELN;
   Alter_To_Choice( R_Num, Fsize)      {Put in spot desired.}
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
FUNCTION More_Rules_Wanted : CHAR;

BEGIN
   Gotoxy(1,24);
   WRITELN;
   Clreol;
   WRITE('Add another rule (Y/N) ? ');
   More_Rules_Wanted:= Get_Valid_Choice_From( Yesno);
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Add_Rule;
{Gets all conditions for rule, then gets action. Then shows rule}
{and checks if user wants to go ahead with  store.  If  so  then}
{stores in rule base and determines priority  user  desires  for}
{rule, setting priority file  entry  accordingly.  Repeats these}
{steps until user indicates that does not want  to  add  another}
{rule to the rule base.                                          }
   VAR Rule_Making : Rules;
       Add_Choice : CHAR;
            R_Num : INTEGER;
            Abort : BOOLEAN;

BEGIN
   REPEAT
      Rule_Making[0].Cond_Num:= 0;
      Creation_Head( Rule_Making);

      Get_All_Conds( Rule_Making, Add_Choice);

      IF ((Conds_In(Rule_Making)<>0) AND (Add_Choice<>'A')) THEN BEGIN

           REPEAT
              Get_New_Action_For( Rule_Making, Abort)
           UNTIL (NOT Abort);

           Show_Rule_And_Check_Ok( Rule_Making, Abort);

           IF (NOT Abort) THEN BEGIN
              Store_In_Rule_Base( Rule_Making, R_Num);
              Find_And_Set_Rule_Priority( R_Num)
           END
      END;

      Add_Choice:= More_Rules_Wanted
   UNTIL (Add_Choice = 'N')
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE List_A_Rule;
    VAR Rule_To_List : Rules;
        Try_Again : CHAR;
             Rnum : INTEGER;

BEGIN
   Menu_Head;
   Get_Valid_Rule_Num( Rnum, Rule_To_List);

   IF (Rnum = 0) THEN
      REPEAT
          Menu_Head;
          Gotoxy(1,24);
          WRITE('That rule not found in rule base. Try another (Y/N) ?');
          Try_Again:= Get_Valid_Choice_From( Yesno);
          IF (Try_Again = 'Y') THEN
              Get_Valid_Rule_Num( Rnum, Rule_To_List)
      UNTIL ((Rnum <> 0) OR (Try_Again = 'N'));

   IF (Rnum <> 0) THEN BEGIN
      WRITELN('Rule #',Rnum,'.');
      Lowvideo;
      WRITELN;
      Show_The_Rule( Rule_To_List);
   END;
   Pause_It
ND;
-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Process_Learn_Options;
{Branching routine for the LEARN module, this passes control}
{to the routine appropriate to the user's menu choice.      }

BEGIN
   CASE Learn_Choice OF
        'A' : Add_Rule;
        'C' : Change_Rule_Action;
        'D' : Delete_Rule;
        'E' : Kill_Cond_In_Rule;
        'H' : Give_Learn_Help;
        'I' : Insert_Condition_In_Rule;
        'L' : List_A_Rule;
        'P' : Alter_Rule_Priority;
        'X' : {Do nothing, exit from learning system wanted.}
   END
END;
{-------------------------------------------------------------------}



{-------------------------------------------------------------------}
PROCEDURE Finalise_Learn_Files;
{Closes learn module files, and re-opens play files}
{if necessary (ie. if closed these on learn entry.)}

BEGIN
    CLOSE( Rbase);                    {Save changes.}
    CLOSE( Pribase);
    ASSIGN( Rbase, 'RULES' + Curbase + '.STA');
    RESET( Rbase);
    ASSIGN( Pribase, 'PRIOR' + Curbase + '.STA');
    RESET( Pribase);
END;
{-------------------------------------------------------------------}


BEGIN                                 {of LEARNER procedure.}
   REPEAT
      Show_Learn_Options;
      Process_Learn_Options
   UNTIL (Learn_Choice = 'X');
   Finalise_Learn_Files;
   Clrscr
END;
{-------------------------------------------------------------------}

END.
```

```
UNIT Ts8;
{This unit contains routines which allow for the listing}
{of rules, conditions, actions, etcetera.                }

Interface

USES Ts1, Dos, Crt;

PROCEDURE Lister;

IMPLEMENTATION

{////////////////////////////////////////////////////////////////////////}
PROCEDURE Lister;
{This routine allows the user to obtain various lists. Lists}
{can be output to terminal or printer, depending  on  wishes}
{of the user, while all selection of  lists  is  menu-based.}
{Lists catered for include listings of  condition and action}
{bases, abbreviated and full listing of rule-base,  priority}
{listings of rules in specific areas, and listing of  single}
{rules.                                                      }
CONST Terminal = TRUE;
      Printer = FALSE;

      Many_Rules = TRUE;
         One_Off = FALSE;


TYPE Str86 = String[86];

VAR Lines_Per_Page, Cur_Line, Cur_Page : BYTE;
                T_Und_Line, P_Und_Line : Str86;
                      Rnam, Anam, Cnam : Str12;
                        List_Rule_Base : FILE OF Rules;
                           List_Choice : CHAR;
                               Headlin : Str80;
                               List_Pr : ARRAY[1..3] OF Int_File;
                                  Pnam : ARRAY[1..3] OF Str12;
                                  Sink : BOOLEAN;



{////////////////////////////////////////////////////////////////////////}
PROCEDURE Init_List;
  VAR I: BYTE;

BEGIN
   Sink:= Terminal;
   Rnam:= 'UNSET';
   List_Choice:= 'X';
   T_Und_Line[0]:= CHR(78);
   P_Und_Line[0]:= CHR(86);
   FOR I:= 1 TO 78 DO
       T_Und_Line[I]:= ' ';
   P_Und_Line[1]:= CHR(27);
   P_Und_Line[2]:= '-';
   P_Und_Line[3]:= CHR(1);
   FOR I:= 4 TO 83 DO
       P_Und_Line[I]:= ' ';
   P_Und_Line[84]:= CHR(27);
   P_Und_Line[85]:= '-';
   P_Und_Line[86]:= CHR(0)
END;
{////////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Give_List_Help;

BEGIN
   Help_Head;
   WRITELN('                    HELP  on  Lister');
   WRITELN;
   Lowvideo;
   WRITELN('The lister module allows the output of different lists to');
   WRITELN('screen or printer. The output sink can be toggled between');
   WRITELN('screen and printer by using the 'S' option in the  Lister');
   WRITELN('menu. The various listing possibilities are :');
   WRITELN('[1] Action listing - lists all actions  that  the  system');
   WRITELN('    caters for.');
   WRITELN('[2] Condition listing - lists all conditions catered for.');
   WRITELN('[3] Rule Listing - within this option you have three sub-');
   WRITELN('    options. You can get a listing of the conditions  and');
   WRITELN('    actions in a single rule with text describing  these,');
   WRITELN('    or a listing of all the rules  in  the  rule-base  in');
   WRITELN('    this  form,  or an  abbreviated listing of rules that');
   WRITELN('    just gives the numbers of conditions and actions  and');
   WRITELN('    not the text for them (a lot shorter than full list).');
   WRITELN('[4] Priority Listing - here a list of rule numbers in the');
   WRITELN('    descending order of their priority is output. You get');
   WRITELN('    the choice to just get a listing for a particular set');
   WRITELN('    of rules - eg. just those concerning opening leads  -');
   WRITELN('    or all rules.                                        ');
   Wait_Till_Read
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Set_Up_Sink;
{Sets page and line counts to initial values  before a}
{listing begins, and sets LINES_PER_PAGE to the amount}
{desirable for the current output sink.                }

BEGIN
   IF (Sink = Terminal) THEN
        Lines_Per_Page:= 20
   ELSE Lines_Per_Page:= 55;
   Cur_Line:= 0;
   Cur_Page:= 1
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Write_Sink;
{Writes out what current output sink is.}

BEGIN
   Normvideo;
   IF (Sink = Terminal) THEN
        WRITE('Terminal')
   ELSE WRITE('Printer ');
   Lowvideo
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Show_List_Options;

BEGIN
   Menu_Head;
   WRITELN('                    Listing Options.       ');
   WRITELN;
   Lowvideo;
   WRITE('Current sink : ');
   Write_Sink;
   WRITELN;
   WRITELN;
   Menu_Write('A =  listing of all Actions available.');
   Menu_Write('C =  listing of all Conditions available.');
   Menu_Write('H =  get Help on listing module.');
   Menu_Write('R =  listing of Rule(s).');
   Menu_Write('P =  listing of rule Priorities.');
   Menu_Write('S =  change current output Sink.');
   Menu_Write('Q =  Quit listing module.')
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Get_List_Choice;
{Gets user's choice of type of list wanted.}

BEGIN
   Gotoxy(1,24);
   Normvideo;
   WRITE('Please enter choice or 'H' for Help : ');
   REPEAT
      List_Choice:= Readkey;
      List_Choice:= Upcase(List_Choice)
   UNTIL (List_Choice IN ['A','C','H','P','Q','R','S']);
   Clrscr
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Write_Head (Headln : Str80);
{Writes the string HEADLN after first paging - clearing}
{screen if terminal output, or sending paging character}
{if printer - ensuring that it is written in 'bold'. ie}
{higher intensity video than used in rest of listing if}
{printing to screed, or with bold escape sequence if to}
{the printer.                                          }
   VAR I : BYTE;

BEGIN
   Clrscr;
   IF (Sink = Printer) THEN BEGIN
      WRITE(Prt,CHR(12));
      WRITE(Prt,CHR(27),'E',Headln,CHR(27),'F');
      FOR I:= 70 DOWNTO LENGTH(Headlin) DO
          WRITE(Prt,' ');
      WRITELN(Prt,'Page ',Cur_Page);
      WRITELN(Prt);
      WRITELN('Listing page ',Cur_Page,' to printer.')
   END
   ELSE BEGIN
      Normvideo;
      WRITELN(Headlin);
      Lowvideo;
      WRITELN
   END
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////////}
PROCEDURE Do_Paging;
{Increments page count,  resets line count, then forces new page and}
{new heading if printer, else prompts user to press key (so ensuring}
{has time to read old page) before resetting page number and placing}
{cursor below heading, clearing all below it.                        }
   VAR I : INTEGER;

BEGIN
   Cur_Page:= Cur_Page + 1;
   Cur_Line:= 0;
   IF (Sink = Printer) THEN
       Write_Head( Headlin)
   ELSE BEGIN
       Gotoxy(1,25);
       WRITE('Press any key to continue.');
       Anykey:= Readkey;
       Gotoxy(70,1);
       WRITELN('Page ',Cur_Page);
       Clear_From(2);
       WRITELN
   END
END;
{////////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////////}
PROCEDURE Line_Write( St_In : Str86);
{Writes the ST_IN string to current sink, incrementing}
{line count and performing paging when needed.        }

BEGIN
   Cur_Line:= Cur_Line + 1;
   IF (Sink = Printer) THEN BEGIN
        WRITE('.');          {So user sees something happening!}
        WRITELN(Prt,St_In)
   END
   ELSE WRITELN(St_In);
   IF (Cur_Line = Lines_Per_Page) THEN
       Do_Paging
END;
{////////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Pretty_Print( Out_Line : Str86; From_Cond : BOOLEAN;
                                                  Num : INTEGER);
{Used for printing of condition and action bases, this starts by}
{writing an underscore line to current  sink,  then  writes  the}
{cond/act number, and then the actual text. A check is  kept  on}
{line count here to ensure that paging is done  is  such  a  way}
{that all three of these lines appear on the same output page.  }
   VAR Title_Line : Str86;
       Temp_Str : Str8;

BEGIN
    Temp_Str:= '';

    IF (Sink = Terminal) THEN BEGIN
         Line_Write(T_Und_Line);
         Normvideo
    END
    ELSE BEGIN
         Line_Write(P_Und_Line);
         Temp_Str:= CHR(27) + 'E'
    END;

    IF From_Cond THEN
         Title_Line:= Temp_Str + 'Condition '
    ELSE Title_Line:= Temp_Str + 'Action ';
    STR(Num,Temp_Str);
    Title_Line:= Title_Line + Temp_Str;
    Line_Write(Title_Line);

    IF (Sink = Printer) THEN
        WRITE(Prt,CHR(27),'F')
    ELSE Lowvideo;

    Line_Write(Out_Line);

    IF ((Sink = Terminal) AND (Cur_Line = 18)) THEN BEGIN
        Line_Write(T_Und_Line);
        Line_Write(' ')           {Take count to 20, ensuring paging. }
    END
    ELSE
       IF (Cur_Line = 54) THEN    {Note must be printer if count > 20.}
          Line_Write(P_Und_Line) {Take count to 55, ensuring paging. }
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Close_Out_List;
{Writes an underscore line at end of list, and calls  DO_PAGING}
{if sink is terminal to ensure user has time to read last page.}
BEGIN
   IF (Sink = Terminal) THEN BEGIN
        Line_Write( T_Und_Line);
        Do_Paging
   END
   ELSE Line_Write( P_Und_Line)
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////////}
PROCEDURE Do_Action_Listing;
{Goes through each entry in the action base text, calling}
{PRETTY_PRINT to write out action number and text.       }
  VAR Out_Line : Str80;
          Cnt : INTEGER;

BEGIN
   RESET( Abase);
   Cnt:= 0;
   Headlin:= 'Listing of Action-Base. ';
   Write_Head(Headlin);
   WHILE (NOT (EOF( Abase))) DO BEGIN
         Cnt:= Cnt + 1;
         READ( Abase,Out_Line);
         Pretty_Print(Out_Line,FALSE,Cnt)
   END;
   Close_Out_List
END;
{////////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////////}
PROCEDURE Do_Condition_Listing;
{Goes through each entry in the condition base text, calling}
{PRETTY_PRINT to write out condition number and text.       }
  VAR Out_Line : Str80;
          Cnt : INTEGER;

BEGIN
   RESET( Cbase);
   Cnt:= 0;
   Headlin:= 'Listing of Condition-Base. ';
   Write_Head(Headlin);
   WHILE (NOT (EOF( Cbase))) DO BEGIN
         Cnt:= Cnt + 1;
         READ( Cbase,Out_Line);
         Pretty_Print(Out_Line,TRUE,Cnt)
   END;
   Close_Out_List
END;
{////////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////////}
PROCEDURE Write_Abbreviated( R_Num : INTEGER; Rule_Got : Rules);
{Writes the rule number, the numbers of the conditions in it, and}
{the number of its action.                                       }
    VAR Condit_Num, I : INTEGER;
            Abrv_Line : Str80;
               Numstr : Str12;

BEGIN
   STR(R_Num, Numstr);
   Abrv_Line:= '#' + Numstr + ' Conds: ';
   FOR I:= 1 TO Conds_In( Rule_Got) DO BEGIN
       Condit_Num:= Rule_Got[I].Cond_Num;
       STR(Condit_Num, Numstr);
       Abrv_Line:= Abrv_Line + Numstr;
       IF (I <> Conds_In( Rule_Got)) THEN
            Abrv_Line:= Abrv_Line + ','
       ELSE Abrv_Line:= Abrv_Line + ' Act: '
   END;
   STR(Rule_Got[20].Cond_Num, Numstr);
   Abrv_Line:= Abrv_Line + Numstr;
   Line_Write(Abrv_Line)
END;
{////////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////////}
PROCEDURE Short_List_Rules;
{Gets each rule in rule base, and calls  WRITE_ABBREVIATED to}
{write out the rule with condition and action numbers instead}
{of cond/act text as would be given in full listing.          }
   VAR Rule_Got : Rules;
          R_Num : INTEGER;

BEGIN
   Headlin:= 'Abbreviated listing of rule base.';
   Write_Head( Headlin);
   R_Num:= 1;
   RESET(Rbase);
   WHILE (NOT (EOF ( Rbase))) DO BEGIN
         READ( Rbase, Rule_Got);
         Write_Abbreviated( R_Num, Rule_Got);
         R_Num:= R_Num + 1
   END;
   IF (Sink = Terminal) THEN Do_Paging
END;
{////////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////////}
FUNCTION Add_Ordinal( Parm_Num : BYTE) : Str80;
   VAR Position : Str12;

BEGIN
   Parm_Num:= Parm_Num - 100;
   STR( Parm_Num, Position);
   IF (Parm_Num > 19) THEN
      Parm_Num:= Parm_Num MOD 20;
   CASE Parm_Num OF
          0 : Position:= Position + 'th ';
          1 : Position:= Position + 'st ';
          2 : Position:= Position + 'nd ';
          3 : Position:= Position + 'rd ';
     4..19 : Position:= Position + 'th '
   END;
   Add_Ordinal:= Position
END;
{////////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////////}
PROCEDURE Add_Parm(VAR Result_Str : Str80; Parm_Num : BYTE);
{Returns RESULT_STR as itself  plus text  corresponding to the}
{PARM_NUM number defining the condition/action parameter used.}
   VAR Temp_Str : Str80;

BEGIN
   STR( Parm_Num, Temp_Str);
   IF (Parm_Num < 101) THEN
        Result_Str:= Result_Str + Temp_Str
   ELSE Result_Str:= Result_Str + Add_Ordinal( Parm_Num)
END;
{////////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
FUNCTION Text_Parm_Version( Str_In : Str80; Cond_In : Condition) : Str80;
{Takes in STR_IN, and returns a string where parameter flags in STR_IN}
{have been replaced by corresponding parameter values while  all  else}
{left unchanged.                                                      }
  VAR Parm_Count, Pos : BYTE;
      Result_Str, Item : Str80;

BEGIN
  Result_Str:= '';
  Pos:= 1;
  Parm_Count:= 0;
  WHILE (NOT (Eo_St(Str_In,Pos))) DO BEGIN
        Get_Item(Item,Pos,Str_In);
        IF (Item[1] = '!') THEN BEGIN
             Parm_Count:= Parm_Count + 1;
             Add_Parm(Result_Str, Cond_In.C_Parms[ Parm_Count])
        END
        ELSE Result_Str:= Result_Str + Item
  END;
  Text_Parm_Version:= Result_Str
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE String_Condition(Cond_In : Condition; VAR Result_Str : Str80);
{Finds the condition text in the condition base, and then uses}
{TEXT_PARM_VERSION to get a string  containing  the  condition}
{base text with parameter flags replaced by appropriate text. }
  VAR Constr : Str80;
      Cnd_Num : INTEGER;

BEGIN
  Cnd_Num:= Cond_In.Cond_Num;
  SEEK( Cbase, (Cnd_Num - 1));
  READ( Cbase, Constr);
  Result_Str:= Text_Parm_Version( Constr, Cond_In)
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
FUNCTION Do_Rule_Console_Paging : BOOLEAN;
{Returns FALSE if user wishes to quit listing of}
{rules to console, else TRUE.                   }

BEGIN
    Gotoxy(1,25);
    WRITE('Press <ESC> to quit, else any key to continue.');
    Anykey:= Readkey;
    IF (Anykey = CHR(27)) THEN
         Do_Rule_Console_Paging:= FALSE
    ELSE BEGIN
         Gotoxy(70,1);
         WRITELN('Page ',Cur_Page);
         Clear_From(2);
         Cur_Line:= 0;
         WRITELN;
         Do_Rule_Console_Paging:= TRUE
    END
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Write_The_Rule( Rnum : INTEGER; Rule_Found : Rules;
                                          VAR Carry_On : BOOLEAN);
{Writes conditions and actions of rule RNUM to current sink. The}
{CARRY_ON parm shows whether call from listing of  just one rule}
{rule or from full base listing - in latter case  escape  option}
{must be given, with CARRY_ON reset accordingly.                 }
   VAR Rnum_Str, Cnt_Str : Str12;
        R_Line, Text_Str : Str80;
           Condit_Number : INTEGER;
                 Cnd_Cnt : BYTE;

BEGIN
   STR( Rnum, Rnum_Str);
   R_Line:= 'CONDITIONS OF RULE #' + Rnum_Str;
   Line_Write(R_Line);
   FOR Cnd_Cnt:= 1 TO Conds_In( Rule_Found) DO BEGIN
       STR( Cnd_Cnt, Cnt_Str);
       String_Condition( Rule_Found[ Cnd_Cnt], Text_Str);
       R_Line:= '[' + Cnt_Str + ']' + Text_Str;
       Line_Write(R_Line)
   END;
   R_Line:= 'ACTION OF RULE #' + Rnum_Str;
   Line_Write(R_Line);
   {$I-}
   SEEK( Abase, (Rule_Found[20].Cond_Num - 1));
   READ( Abase, Text_Str);
   {$I+}
   IF ioresult <> 0 THEN
      WRITELN('!!! error reading action ', rule_found[20].cond_num);
   Text_Str:= Text_Parm_Version( Text_Str, Rule_Found[20]);
   Line_Write( Text_Str);
   IF (Sink = Terminal) THEN
        IF Carry_On THEN
             Carry_On:= Do_Rule_Console_Paging
        ELSE Do_Paging
   ELSE BEGIN
        Line_Write(P_Und_Line);
        IF (Cur_Line > 40) THEN
            Do_Paging
   END
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Get_Legal_Integer( VAR Intval : INTEGER; Colst, Rowst : BYTE);
{Used to allow user to make mistakes in integer entry without causing}
{a run-time error. Integers read in as strings, and the VAL  function}
{then applied. If ERRFLAG then shows integer not  syntactically okay,}
{then send error message and return to the COLST and ROWST  specified}
{in arguments for user to re-try.                                    }
    VAR Errflag : INTEGER;
          Instr : Str12;

BEGIN
   REPEAT
        Gotoxy(Colst,Rowst);
        Clreol;
        READLN(Instr);
        VAL(Instr,Intval,Errflag);
        IF (Errflag <> 0) THEN BEGIN
            Gotoxy(1,Rowst);
            WRITE('Invalid integer - retry : ');
            Clreol
        END
   UNTIL (Errflag = 0)
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Get_Valid_Rule_Num( VAR R_Num : INTEGER; VAR Rule_Got : Rules);
{Gets integer from user and then tests that such a rule does}
{in fact exist, sending error message if not.                }

BEGIN
   Gotoxy(1,24);
   Clreol;
   WRITE('Please enter rule number : ');
   Get_Legal_Integer(R_Num,28,24);

   {$I-}
   SEEK(Rbase, (R_Num - 1));
   IF (Ioresult <> 0) THEN R_Num:= 0
   ELSE BEGIN
        READ(Rbase,Rule_Got);
        IF (Ioresult <> 0) THEN R_Num:= 0
   END;
   {$I+}

   Gotoxy(1,24);
   Clreol
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE List_A_Rule;
{Prompts user until a valid rule number received, and then lists}
{that rule to the current output sink.                          }
   VAR Try_Again : CHAR;
       Who_Calls : BOOLEAN;
        Rule_Got : Rules;
           R_Num : INTEGER;

BEGIN
   Menu_Head;
   Who_Calls:= One_Off;    {Shows WRITE_THE_RULE that only one...}
                           {rule being listed here.               }
   Try_Again:= 'Y';
   Get_Valid_Rule_Num( R_Num, Rule_Got);
   IF (R_Num = 0) THEN
      REPEAT
          Menu_Head;
          Gotoxy(1,24);
          WRITE('That rule not found in rule base. Try another (Y/N) ?');
          Try_Again:= Get_Valid_Choice_From( Yesno);
          IF (Try_Again = 'Y') THEN
              Get_Valid_Rule_Num( R_Num, Rule_Got)
      UNTIL ((R_Num <> 0) OR (Try_Again = 'N'));
   Menu_Head;
   Lowvideo;
   IF (Try_Again <> 'N') THEN
       Write_The_Rule( R_Num, Rule_Got, Who_Calls)
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Full_Rule_List;
{Repeatedly gets rules from rule-base and lists them to current}
{output sink using WRITE_THE_RULE until all rules written.      }
   VAR Rule_Got : Rules;
         R_Num : INTEGER;
         Go_On : BOOLEAN;

BEGIN
   Headlin:= 'Full listing of current rule base.';
   Write_Head( Headlin);
   R_Num:= 1;
   SEEK( Rbase, 0);
   Go_On:= TRUE;                          {User abort flag.          }
   WHILE ((NOT (EOF (Rbase))) AND Go_On) DO BEGIN
         READ( Rbase, Rule_Got);
         Write_The_Rule( R_Num, Rule_Got, Go_On);
         R_Num:= R_Num + 1
   END
END;
{////////////////////////////////////////////////////////////////////}


{////////////////////////////////////////////////////////////////////}
PROCEDURE Do_Rule_Listing;
{Gives user different rule listing options, gets choice, and}
{then passes control to the appropriate listing routine.    }
   VAR R_Choice : CHAR;
         Outstr : Str80;

BEGIN
   Menu_Head;
   WRITE('     Rule  Listing  Menu.                Current sink : ');
   Write_Sink;
   WRITELN;
   WRITELN;
   Outstr:=
    'A =  Abbreviated listing of rule-base, condition/action numbers only.';
   Menu_Write(Outstr);
   Menu_Write('F =  Full listing of rule-base, condition/action text.');
   Menu_Write('P =  full listing of a Particular rule.');
   Menu_Write('S =  change current output Sink.');
   Gotoxy(1,25);
   WRITE('Please enter choice : ');
   R_Choice:= Get_Valid_Choice_From(['A','F','P','S']);

   CASE R_Choice OF
        'A' : Short_List_Rules;
        'F' : Full_Rule_List;
        'P' : List_A_Rule;
        'S' : Sink:= NOT(Sink)
   END
END;
{////////////////////////////////////////////////////////////////////}


{////////////////////////////////////////////////////////////////////}
PROCEDURE Show_Pri_Options;
{Displays the menu of choices for a priority listing.}

BEGIN
   Menu_Head;
   WRITE('     Priority Listing Options.          Current sink : ');
   Write_Sink;
   WRITELN;
   WRITELN;
   Menu_Write('A =  get All rule priorities.');
   Menu_Write('Q =  Quit priority listing menu.');
   Menu_Write('S =  change current output Sink.');
   Gotoxy(1,24);
   WRITE('Please enter choice : ');
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Get_Pri_Choice( VAR Pkey, Full_List : CHAR);
{Gets choice from user, and if choice for a list  of  rules}
{to be printed then determines whether condition and action}
{text wanted along with rule, or just  rule  number. Passes}
{user's decisions back via the two character arguments.    }

BEGIN
   Pkey:= Get_Valid_Choice_From(['A','Q','S']);
   IF ((Pkey <> 'Q') AND (Pkey <> 'S')) THEN BEGIN
       Gotoxy(1,24);
       Clreol;
       WRITE('Full listing of rules, or rule Numbers only (F/N) ? ');
       Full_List:= Get_Valid_Choice_From(['F','N'])
   END
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Priority_Print( Full_List : CHAR);
{Repeatedly gets rule numbers from the relevant  priority file}
{until end of priority file reached. For each rule number read}
{the corresponding rule is found in the rule-base  and  output}
{to the current sink.                                         }
    VAR Rule_There : Rules;
          Rule_Num : INTEGER;
          Out_Line : Str80;
             Go_On : BOOLEAN;

BEGIN
   Go_On:= TRUE;                              {User list abort flag.    }
   Headlin:= 'Priorities of trail typing rules';
   Headlin:= Headlin + ' - descending priority order.';
   Write_Head(Headlin);
   SEEK( Pribase, 0);
   WHILE ((NOT(EOF( Pribase))) AND Go_On) DO BEGIN
         READ( Pribase, Rule_Num);
         IF (Full_List = 'N') THEN BEGIN   {Abbreviated rules wanted.}
             STR(Rule_Num, Out_Line);
             Out_Line:= 'Rule ' + Out_Line;
             Line_Write(Out_Line)
         END
         ELSE BEGIN                          {Full text wanted.        }
             SEEK( Rbase, (Rule_Num - 1));
             READ( Rbase, Rule_There);
             Write_The_Rule(Rule_Num, Rule_There, Go_On)
         END
   END;
   IF ((Sink = Terminal) AND (Full_List = 'N')) THEN
       Do_Paging
END;
{////////////////////////////////////////////////////////////////////}
```

```
{////////////////////////////////////////////////////////////////////}
PROCEDURE Give_Priority_List_Wanted( Pkey, Full_List : CHAR);
{Calls priority print to print out appropriate group(s) of rules}
{if the PKEY choice of user indicates a listing wanted. If not a}
{listing wanted then either user requested a  Quit from menu, in}
{which case no action taken, or wanted  sink  changed.  In  this}
{latter case the SINK variable is reset, and then SET_UP_SINK is}
{called to ensure that max line count, etc, properly reset.     }

BEGIN
   CASE Pkey OF
        'A' : BEGIN
                 Priority_Print( Full_List);
                 Cur_Page:= Cur_Page + 1;
              END;
        'Q' : {Do nothing};
        'S' : BEGIN
                 Sink:= NOT(Sink);
                 Set_Up_Sink
              END
   END
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Do_Priority_Listing;
{Shows priority listing options, and gets users choice to determine}
{which priority file(s) will be used. Then determines whether  user}
{wishes rules to be  printed in full, (with condition/action text),}
{or not, and passes control and information received from  user  to}
{the PRIORITY_PRINT routine.                                       }
    VAR Pkey, Full_List : CHAR;

BEGIN
  REPEAT
     Show_Pri_Options;
     Get_Pri_Choice( Pkey, Full_List);
     Give_Priority_List_Wanted( Pkey, Full_List);
     Clrscr;
  UNTIL (Pkey = 'Q')
END;
{////////////////////////////////////////////////////////////////////}



{////////////////////////////////////////////////////////////////////}
PROCEDURE Do_List_Choice_Action;
{Branching routine which passes control to the listing}
{routine indicated by value of LIST_CHOICE.          }

BEGIN
  CASE List_Choice OF
       'A' : Do_Action_Listing;
       'C' : Do_Condition_Listing;
       'H' : Give_List_Help;
       'R' : Do_Rule_Listing;
       'P' : Do_Priority_Listing;
       'S' : Sink:= NOT Sink;
       'Q' : {Do nothing}
  END;
  Set_Up_Sink
END;
{////////////////////////////////////////////////////////////////////}
```

```
BEGIN                          {Of LISTER procedure.}
   Init_List;
   Set_Up_Sink;
   REPEAT
       Show_List_Options;
       Get_List_Choice;
       Do_List_Choice_Action
   UNTIL (List_Choice = 'Q');
END;
{////////////////////////////////////////////////////////////////////////}


END.
```

```
UNIT Ts9;
{This unit contains routines for trail access, as well as routines}
{relating to user queries regarding trail classification.         }

Interface

USES Ts1, Ts2, Ts3, Ts4, Ts5, Ts6, Ts7, Ts8, Dos, Crt;

PROCEDURE Get_Next_Trail;
PROCEDURE Get_Trail;
PROCEDURE Get_Next_Trail_Type (     Trail_Type    : BYTE;
                                 VAR Successful    : BOOLEAN);
PROCEDURE Change_Curname( VAR Base_Used : BYTE);
PROCEDURE Send_Crash_Message_And_Abort;
PROCEDURE Show_Why_Nots;
PROCEDURE Query;


IMPLEMENTATION


{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
PROCEDURE Get_Next_Trail;
 VAR   Cnt, I  : INTEGER;
     Real_Gap : REAL;
         Sval : BYTE;
          Gap : INTEGER;
         Cpos : INTEGER;
         Posi : LONGINT;

BEGIN
  Prev_Rec:= Hrec;
  IF (NOT (EOF (Trail_Head))) THEN BEGIN
      READ( Trail_Head, Hrec);
      Real_Gap:= (Hrec.Sample_Range[1] - Hrec.Sample_Range[0]) + 1;
      Cnt:= 1;
      IF ((Real_Gap > 10000.0) OR (Real_Gap < 3.0)) THEN BEGIN
          Trail_Array [0] := 0;
          Garbage_Flag := TRUE
      END
      ELSE BEGIN
         Gap:= TRUNC( Real_Gap);
         Posi:= ROUND( Hrec.Sample_Range[0]);
         SEEK( Trail_Sample, Posi);
         Garbage_Flag:= FALSE;
         FOR I:= 1 TO Gap DO BEGIN
            READ( Trail_Sample, Sval);
            IF (Sval > 60) THEN
               Garbage_Flag:= TRUE;
            Trail_Array[Cnt]:= Sval;
            Cnt:= Cnt + 1
         END;
         Cnt:= Cnt - 1;
         Trail_Array[0]:= Cnt;
      END
   END;
END;
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
```

```
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
PROCEDURE Get_Trail;

VAR  Trail_Str                 : Str12;
     Sval                      : BYTE;
     Real_Gap                  : REAL;
     Err, Gap                  : INTEGER;
     Trail_Number, Cnt, I      : INTEGER;
     Posi                      : LONGINT;

BEGIN
  REPEAT
    Gotoxy (1, 24);
    Clreol;
    Gotoxy (1, 24);
    WRITE ('Enter trail number between ', Start_Trail);
    WRITE (' and ', End_trail, '  -   ');
    READLN(Trail_Str);
    VAL (Trail_Str, Trail_Number, Err)
  UNTIL (Trail_Number >= Start_Trail) AND (Trail_Number <= End_Trail) AND
        (Err = 0);
  SEEK( Trail_Head,0);
  READ (Trail_Head, Hrec);
  Prev_Rec:= Hrec;    {Yes, you DO need this - might not enter WHILE loop.}
  WHILE (NOT (EOF (Trail_Head))) AND (Hrec.Trail_Num <> Trail_Number) DO
   BEGIN
      Prev_Rec:= Hrec;
      READ (Trail_Head, Hrec);
   END;
  IF Hrec.Trail_Num <> Trail_Number THEN
   BEGIN
     Gotoxy (1, 24);
     WRITE ('This trail is not in the file');
     Clreol;
     Trail_Array [0] := 0;
     Garbage_Flag := TRUE
   END
  ELSE
   BEGIN
     Cnt:= 1;
     Real_Gap:= TRUNC(Hrec.Sample_Range[1] - Hrec.Sample_Range[0] + 1);
     IF ((Real_Gap > 10000.0) OR (Real_Gap < 3.0)) THEN
      BEGIN
        Trail_Array [0] := 0;
        Garbage_Flag := TRUE
      END
     ELSE
      BEGIN
        Gap:= TRUNC( Real_Gap);
        Posi:= ROUND( Hrec.Sample_Range[0]);
        SEEK( Trail_Sample, Posi);
        Garbage_Flag:= FALSE;
        FOR I:= 1 TO Gap DO BEGIN
            READ( Trail_Sample, Sval);
            IF (Sval > 60) THEN           {Value over -80 dBm.}
               Garbage_Flag:= TRUE;
            Trail_Array[Cnt]:= Sval;
            Cnt:= Cnt + 1
        END;
        Cnt:= Cnt - 1;
        Trail_Array[0]:= Cnt;
      END
   END
END;
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
```

```
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
PROCEDURE Get_Next_Trail_Type (     Trail_Type    : BYTE;
                                    VAR Successful    : BOOLEAN);

   VAR Cnt, I : INTEGER;
         Sval : BYTE;
     Real_Gap : REAL;
          Gap : INTEGER;
         Cpos : INTEGER;
         Posi : LONGINT;

BEGIN
  IF NOT (EOF (Trail_Head)) THEN
   BEGIN
     Prev_Rec:= Hrec;
     REPEAT
       READ (Trail_Head, Hrec)
     UNTIL EOF (Trail_Head) OR (Hrec.Trail_Type = Trail_Type);
     Successful := Hrec.Trail_Type = Trail_Type;
     IF Successful THEN
      BEGIN
        Cnt:= 1;
        Real_Gap:= TRUNC(Hrec.Sample_Range[1] - Hrec.Sample_Range[0] + 1);
        IF ((Real_Gap > 10000.0) OR (Real_Gap < 1.0)) THEN
         BEGIN
           Trail_Array [0] := 0;
           Garbage_Flag := TRUE
         END
        ELSE
         BEGIN
           Gap:= TRUNC( Real_Gap);
           Posi:= ROUND( Hrec.Sample_Range[0]);
           SEEK( Trail_Sample, Posi);
           Garbage_Flag:= FALSE;
           FOR I:= 1 TO Gap DO BEGIN
               READ( Trail_Sample, Sval);
               IF (Sval > 60) THEN
                   Garbage_Flag:= TRUE;
               Trail_Array[Cnt]:= Sval;
               Cnt:= Cnt + 1
           END;
           Cnt:= Cnt - 1;
           Trail_Array[0]:= Cnt
         END
      END
   END
  ELSE
    Successful := FALSE;
END;
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
```

```
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
PROCEDURE Change_Curname( VAR Base_Used : BYTE);
    VAR Ch_Choice : CHAR;
          Old_Dir : String;
             Badf : BOOLEAN;
             Hnum : INTEGER;

BEGIN
  Clear_From( 23);
  Gotoxy( 1,24);
  IF (Curname <> 'UNKNOWN') THEN BEGIN
      CLOSE( Trail_Head);
      CLOSE( Trail_Sample);
      Curname:= 'UNKNOWN';
  END;
  WRITE('Select file from Table or Define it specifically (T/D) : ');
  Ch_Choice:= Get_Valid_Choice_From( ['T','D',Esc]);
  CASE Ch_Choice OF
       Esc : Exit;
       'D' : BEGIN
               {$I-}
               REPEAT
                  Gotoxy( 1, 24);
                  Clreol;
                  Gotoxy(1,24);
                  WRITE('Please enter new trail file name : ');
                  READ( Curname);
                  Getdir( 0, Old_Dir);
                  Chdir( Path);
                  ASSIGN( Trail_Head, Curname);
                  RESET( Trail_Head);
                  Badf:= (Ioresult <> 0);
                  Curname[1]:= 'S';
                  ASSIGN( Trail_Sample, Curname);
                  RESET( Trail_Sample);
                  Chdir( Old_Dir);
               UNTIL ((NOT Badf) AND (Ioresult = 0));
               {$I+}
             END;
       'T' : BEGIN
               Fill_File_Table;
               List_File_Table_To_Screen;
               WRITELN;
               Hnum:= Get_User_Int( 'Please enter choice :', 1,
                                          Number_Files_In_Table);
               IF Exitting THEN BEGIN
                  Exitting:= FALSE;
                  Exit;
               END;
               Curname:= File_Table[ Hnum];
               Getdir( 0, Old_Dir);
               Chdir( Path);
               ASSIGN( Trail_Head, Curname);
               RESET( Trail_Head);
               Curname[1]:= 'S';
               ASSIGN( Trail_Sample, Curname);
               RESET( Trail_Sample);
               Chdir( Old_Dir);
             END;
  END;
  READ (Trail_Head, Hrec);
  Start_Trail := Hrec.Trail_Num;
  Base_Used:= Hrec.Useful_Time[ Plus6,1];
  IF (Base_Used > 100) THEN
     Base_Used:= 0;
  Prev_Rec:= Hrec;
  SEEK (Trail_Head, Filesize (Trail_Head) - 1);
  READ (Trail_Head, Hrec);
  End_Trail := Hrec.Trail_Num;
  Hrec:= Prev_Rec;       {First guy in file must have self as PREV_REC.}
  SEEK(Trail_Head,0);
  Snam := Curname; Curname[1]:= 'H'; Hnam := Curname
END;
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
```

```
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}
PROCEDURE Send_Crash_Message_And_Abort;
{Called if can't find any triggered rule.}

BEGIN
   Alarm(Leave_Cursor);
   Clrscr;
   Normvideo;
   WRITELN('           !!! RULE-BASE CRASH !!!');
   WRITELN;
   WRITELN('Typing aborted as no rule could be fired.');
   WRITELN;
   WRITELN('The 'catch-all' rule at the end of the priority file has been');
   WRITELN('deleted. You MUST ensure that SOME rule holds at all times.');
   WRITELN;
   Play_Choice:= 'X'                         {Force program abortion.}
END;
{''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''}



{????????????????????????????????????????????????????????????????????????????}
PROCEDURE Show_Why_Nots;
   VAR Cond_In_Rule : BYTE;
         I, Badrnum : INTEGER;
           Badrule : Rules;
            Choyce : CHAR;
             Do_It : BOOLEAN;

BEGIN
    Clear_From( 8);
    IF (Test_Failures = 0) THEN
           WRITELN('No failed rules.')
    ELSE
      FOR I:= 1 TO Test_Failures DO BEGIN
           Badrnum:= Failures[I].Rnumber;
           Cond_In_Rule:= Failures[I].Cnumber;
           SEEK( Rbase, Badrnum - 1);
           READ( Rbase, Badrule);
           WRITE('Not typed as ');
           Show_Whynot_Act( Badrule[ 20]);
           WRITELN(' Failed test was : ');
           WRITE(' ');
           Show_Cond( Badrule[ Cond_In_Rule]);
           WRITELN;
           Gotoxy(1,15);
           WRITE('Press ''C'' to continue, ''P'' for Precise numbers,');
           WRITE(' or <Esc> to abort : ');
           Choyce:= Get_Valid_Choice_From(['C','P',Esc]);
           CASE Choyce OF
                Esc : BEGIN
                        Clear_From( 8);
                        Exit;
                      END;
                'C' : Clear_From( 8);
                'P' : BEGIN
                        Clear_From( 15);
                        User_Wants_Actual_Condition_Value:= TRUE;
                        Do_It:= Condition_Checker( Badrule[ Cond_In_Rule]);
                        User_Wants_Actual_Condition_Value:= FALSE;
                        WRITELN;
                        WRITE('Press any key to continue...');
                        Anykey:= Readkey;
                        Clear_From( 8);
                      END;
           END;
      END;
   Gotoxy( 1,24);
   WRITE('Press any key to continue...');
   Anykey:= Readkey;
   Clear_From( 8);
END;
{????????????????????????????????????????????????????????????????????????????}
```

```
{??????????????????????????????????????????????????????????????????????}
PROCEDURE Query;
{This routine displays to the user the last rule that}
{was fired in response to the user's 'Why' question. }


{??????????????????????????????????????????????????????????????????????}
PROCEDURE Init_Why_Screen(R_Num : BYTE);
      VAR I : BYTE;

BEGIN
   Clear_From( 8);
   Normvideo;
   WRITELN('                              Answer to WHY question.');
   WRITE('Rule ',R_Num,' caused ');
   WRITELN(' this typing to be made. Conditions fired (true) were :');
   Lowvideo;
   FOR I:= 1 TO 80 DO WRITE(':');
END;
{??????????????????????????????????????????????????????????????????????}


{??????????????????????????????????????????????????????????????????????}
PROCEDURE Show_Query_Rule( Rule_In : Rules);
{Writes conditions and actions in the rule, using utilities}
{SHOW_COND and SHOW_ACT which ensure  that  parameter flags}
{in text replaced by text corresponding to  the  values  of}
{those parameters in the rule.                             }
  VAR  Num_Conds, I : BYTE;
          Whatcond : Condition;


BEGIN
  Num_Conds:= Rule_In[0].Cond_Num;
  FOR I:= 1 TO Num_Conds DO BEGIN
      Whatcond:= Rule_In[I];
      Show_Cond( Whatcond);
  END;
  Normvideo;
  WRITELN('Action (typing) of rule was : ');
  Lowvideo;
  Show_Act( Rule_In[20])
END;
{??????????????????????????????????????????????????????????????????????}



{??????????????????????????????????????????????????????????????????????}
PROCEDURE End_Why_Screen;
  VAR Whynot : CHAR;

BEGIN
   WRITELN;
   WRITE('Want to see why other rules tested failed? (Y/N) ');
   Whynot:= Get_Valid_Choice_From( [ 'Y', 'N']);
   IF (Whynot = 'Y') THEN
        Show_Why_Nots;
END;
{??????????????????????????????????????????????????????????????????????}



BEGIN                              {of QUERY routine.}
   Init_Why_Screen( First_Choice);
   Show_Query_Rule( First_Fired);
   End_Why_Screen;
END;
{??????????????????????????????????????????????????????????????????????}

END.
```

```
UNIT Ts10;
{This unit contains routines which implement the screen display}
{of meteor trail reflections.                                   }

Interface

USES Printer, Dos, Crt, Graph, Ts1;

PROCEDURE Display_Trail( Caller : BOOLEAN);

IMPLEMENTATION


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Display_Trail;
   VAR Systm, Day_Of, Hour_Of, Year_Of : Str12;
                            Trail_Nam : Str40;
                         Trayl_Num, I : INTEGER;


CONST Leftaxval = 73;
       Endaxval = 575;

       Top_Ax1 = 8;
       Bot_Ax1 = 188;
       Top_Ax2 = 222;
       Bot_Ax2 = 282;

      Leftmain = 2;
       Topmain = 2;
     Rightmain = 638;
       Botmain = 328;

   Max_Ax2_Ms = 2500;

       Max_Dbm = -80;
       Min_Dbm = -140;

        Offset = 4;
     Magnifier = 3;


VAR  Anykey : CHAR;
     Trailclass, Trailname : Str80;
     Noise, Max_Ax1_Ms : INTEGER;
     In_Char : CHAR;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_Trails (Shape_View, Real_View : Data_Array;
                       Noise                 : INTEGER;
                       Max_Ax1_Ms            : INTEGER;
                       Trailname             : Str80);


  VAR Step_Ms : INTEGER;
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Show_Rise_And_Fall_Positions;
  VAR Rise_Ms, Fall_Ms, Rise_Dbm, Fall_Dbm : INTEGER;

     {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
     PROCEDURE Label_Rise_And_Fall_Lines;

     BEGIN
         Settextstyle( Smallfont, Vertdir, 4);
         IF (Risepos = Fallpos) THEN BEGIN
             Moveto( Leftaxval + Rise_Ms, Bot_Ax1 - Rise_Dbm - 90);
             Outtext('Rise/fall spot');
         END
         ELSE BEGIN
             Moveto( Leftaxval + Rise_Ms, Bot_Ax1 - Rise_Dbm - 60);
             Outtext('End rise');
             Moveto( Leftaxval + Fall_Ms, Bot_Ax1 - Fall_Dbm - 70);
             Outtext('Start fall');
         END
      END;
     {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

BEGIN
   Rise_Ms:= ROUND( 500.0 * Risepos/Real_View[0]);
   Fall_Ms:= ROUND( 500.0 * Fallpos/Real_View[0]);
   Rise_Dbm:= Shape_View[ Rise_Ms];
   Fall_Dbm:= Shape_View[ Fall_Ms];
   Setlinestyle( Dottedln, 0, Normwidth);
   Moveto( Leftaxval + Rise_Ms, Bot_Ax1 - Rise_Dbm + 6);
   Lineto( Leftaxval + Rise_Ms, Bot_Ax1 - Rise_Dbm - 6);
   Moveto( Leftaxval + Fall_Ms, Bot_Ax1 - Fall_Dbm + 6);
   Lineto( Leftaxval + Fall_Ms, Bot_Ax1 - Fall_Dbm - 6);
   Label_Rise_And_Fall_Lines;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}



{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Init_Graph;
   VAR Tempstr : String;

BEGIN
   Graphdriver:= Detect;
   Initgraph( Graphdriver, Graphmode, 'C:DRIVERS');
   Errorcode:= Graphresult;
   IF (Errorcode <> Grok) THEN BEGIN
      WRITELN('Graphics problem - ',Grapherrormsg( Errorcode));
      WRITELN('Program aborted');
      Halt(1);
   END
   ELSE BEGIN
      Setgraphmode( Graphmode);
      Settextstyle( Smallfont, Horizdir, 6);
      IF Real_View[0] > 550 THEN  Real_View[0] := 550;
      IF Hrec.Trail_Type = 255 THEN
         Trailclass:= 'Trail not yet classified'
      ELSE
        BEGIN
         SEEK (Abase, Hrec.Trail_Type - 1);
         READ (Abase, Trailclass);
         DELETE( Trailclass, 1, 15);
        END;
  END;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
CONST Xmaxglb = 89;                 { Number of BYTES -1 in one screen line }
      Xscreenmaxglb = 643;         { Number of PIXELS -1 in one screen line }
      Ymaxglb = 343;                     { Number of lines -1 on the screen }

VAR Colorglb : BYTE;


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
FUNCTION Pd(x,y:INTEGER):BOOLEAN;   { Return true if the color of the point at
                                      X,Y matches ColorGlb }
  BEGIN
    Pd:= (Getpixel( X,Y) <> Black)   {NB! - this replacement of the original}
                                     {code in Turbo 3.0 GrafBox is VITAL for}
                                     {the routine to work with version 4.0. }
  END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE hardcopy(inverse:BOOLEAN;mode:BYTE); { EPSON }
  VAR i,j,top:INTEGER;
      Colorloc,Printbyte:BYTE;

  PROCEDURE doline(top:INTEGER);
     VAR M : INTEGER;

    FUNCTION Constructbyte(j,i:INTEGER):BYTE;
      CONST Bits:ARRAY [0..7] OF BYTE=(128,64,32,16,8,4,2,1);
      VAR Cbyte,k:BYTE;
      BEGIN
        i:=i shl 3;
        Cbyte:=0;
        FOR k:=0 TO top DO
          IF Pd(j,i+k) THEN Cbyte:=Cbyte OR Bits[k];
        Constructbyte:=Cbyte;
      END;

    BEGIN              {Of DOLINE}
      IF mode=1 THEN WRITE(lst,^['L')
      ELSE WRITE(lst,^['*',CHR(mode));
      WRITE(lst,CHR(lo(Xscreenmaxglb+1)),CHR(Hi(Xscreenmaxglb+1)));
      FOR M:=0 TO Xscreenmaxglb DO
       BEGIN
        Printbyte:=Constructbyte(M,i);
        IF inverse THEN Printbyte:=NOT Printbyte;
        WRITE(lst,CHR(Printbyte));
       END;
      IF mode<>4 THEN WRITELN(lst);
    END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

BEGIN                   {Of HardCopy Procedure}
    top:=7;
    Colorloc:=Colorglb;
    Colorglb:=255;
    mode:=mode AND 7;
    IF (mode=5) OR (mode=0) THEN mode:=4;
    WRITE(lst,^['3'#24);
    FOR i:=0 TO ((Ymaxglb+1) shr 3)-1 DO doline(7);
    i:=((Ymaxglb+1) shr 3);
    IF (Ymaxglb+1) AND 7<>0 THEN
      doline((Ymaxglb+1) AND 7);
    WRITELN(lst,^['2');
    Colorglb:=Colorloc;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE End_Graph;
   VAR Print_Char : CHAR;

BEGIN
  Print_Char:= Readkey;
  IF Upcase (Print_Char) = 'P' THEN
      Hardcopy( FALSE, 1);
  Restorecrtmode;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}



{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_Square( Y1, X1, Y2, X2 : INTEGER);

BEGIN
   Line( Y1,X1,Y2,X1);
   Line( Y1, X2, Y1, X1);
   Line( Y1, X2, Y2, X2);
   Line( Y2, X1, Y2, X2);
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_Axes;

   PROCEDURE Draw_Ax1;
      VAR Row, Col : INTEGER;

   BEGIN
     FOR Col:= 0 TO 3 DO
         Line( Leftaxval + Col, Top_Ax1, Leftaxval + Col, Bot_Ax1);
     FOR Row:= 0 TO 3 DO
         Line( Leftaxval, Bot_Ax1 + Row, Endaxval, Bot_Ax1 + Row);
   END;


   PROCEDURE Draw_Ax2;
      VAR Row, Col : INTEGER;

   BEGIN
     FOR Col:= 0 TO 3 DO
         Line( Leftaxval + Col, Top_Ax2, Leftaxval + Col, Bot_Ax2);
     FOR Row:= 0 TO 3 DO
         Line( Leftaxval, Bot_Ax2 + Row, Endaxval, Bot_Ax2 + Row);
   END;


BEGIN
   Draw_Ax1;
   Draw_Ax2;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Write_Name_And_Class;
   VAR Title : String;

BEGIN
  Title:= Trailname;
  (* IF (Caller = Typing_Call) THEN *)
     Title:= Title + ' - ' + Trailclass;
  Settextstyle( Smallfont, Horizdir, 5);
  Moveto( 30, 334);
  Outtext( Title);
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Number_Axes( Max_Ax1_Ms : INTEGER);
  VAR Step_Db, Step_Ms : INTEGER;

   PROCEDURE Number_Ax1( Max_Ax1_Ms : INTEGER);
      VAR Numstr : Str80;
           Step_Ms, Off, Spots: INTEGER;

   BEGIN
      FOR Spots:= 0 TO 6 DO BEGIN   {Column axis - 'Y'}
        Moveto( Leftaxval - 33, ((Bot_Ax1 - Offset) - (Spots * Magnifier * 10)));
         STR( 0 - 140 + (Spots * 10), Numstr);
         Outtext( Numstr)
      END;
      Step_Ms:= ROUND (Max_Ax1_Ms/10.0);
      FOR Spots:= 0 TO 10 DO BEGIN   {Row axis - 'X'}
         Moveto( (Leftaxval + (Spots * 50)), Bot_Ax1 + 1);
         Outtext('|');
         IF ((Spots * Step_Ms) < 10) THEN
              Off:= 0
         ELSE
            IF ((Spots * Step_Ms) < 100) THEN
                 Off:= 4
            ELSE
               IF ((Spots * Step_Ms) < 1000) THEN
                    Off:= 8
               ELSE Off:= 12;
         Moveto( (Leftaxval + (Spots * 50)- Off), Bot_Ax1 + 10);
         STR( Spots * Step_Ms, Numstr);
         Outtext( Numstr)
      END;
  END;

  PROCEDURE Number_Ax2;
     VAR Numstr : Str80;
           Step_Ms, Off, Spots: INTEGER;

   BEGIN
      FOR Spots:= 0 TO 3 DO BEGIN
         Moveto( Leftaxval - 33, ((Bot_Ax2 - Offset) - (Spots * 20)));
         STR( 0 - 140 + (Spots * 20), Numstr);
         Outtext( Numstr)
      END;
      Step_Ms:= Max_Ax2_Ms DIV 10;
      FOR Spots:= 0 TO 10 DO BEGIN   {Row axis - 'X'}
         Moveto( (Leftaxval + (Spots * 50)), Bot_Ax2 + 1);
         Outtext('|');
         IF ((Spots * Step_Ms) < 10) THEN
              Off:= 0
         ELSE
            IF ((Spots * Step_Ms) < 100) THEN
                 Off:= 4
            ELSE
               IF ((Spots * Step_Ms) < 1000) THEN
                    Off:= 8
               ELSE Off:= 12;
         Moveto( (Leftaxval + (Spots * 50) - Off), Bot_Ax2 + 10);
         STR( Spots * Step_Ms, Numstr);
         Outtext( Numstr)
      END;
  END;


BEGIN
   Settextstyle( Smallfont, Horizdir, 4);
   Step_Db:= 10;
   Step_Ms:= ROUND (Max_Ax1_Ms/10.0);
   Number_Ax1( Max_Ax1_Ms);
   Step_Db:= 60;
   Step_Ms:= 250;
   Number_Ax2;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Label_Axes;

BEGIN
   Settextstyle( Sansseriffont, Horizdir, 1);
   Moveto( Leftmain + Offset, Bot_Ax1 - 60);
   Outtext( 'dBm');
   Moveto( Leftmain + Offset, Bot_Ax2 - 30);
   Outtext( 'dBm');
   Settextstyle( Smallfont, Horizdir, 4);
   Moveto( Endaxval + 18, Bot_Ax1 + 10);
   Outtext( 'ms');
   Moveto( Endaxval + 18, Bot_Ax2 + 10);
   Outtext( 'ms');
   Settextstyle( Smallfont, Horizdir, 6);
   Moveto( Leftmain + 200, Bot_Ax1 + 20);
   Outtext('''Shape-oriented'' view');
   Moveto( Leftmain + 250, Bot_Ax2 + 20);
   Outtext('''True'' view');
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_Noise;

BEGIN
   Setlinestyle( Dashedln, 0, Normwidth);
   Settextstyle( Smallfont, Horizdir, 4);
   Moveto( Leftaxval, Bot_Ax1 - ((Noise + 140) * Magnifier));
   Lineto( Endaxval, Bot_Ax1 - ((Noise + 140) * Magnifier));
   Moveto( Endaxval, Bot_Ax1 - ((Noise + 140) * Magnifier) - 5);
   Outtext('  Noise');
   Moveto( Leftaxval, Bot_Ax2 - (Noise + 140));
   Lineto(  Endaxval, Bot_Ax2 - (Noise + 140));
   Moveto( Endaxval, Bot_Ax2 - (Noise + 140) - 5);
   Outtext('  Noise');
   Setlinestyle( Solidln, 0, Normwidth);
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}



{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_Graphs( Max_Ax1_Ms : INTEGER);

{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_G1;
  VAR Pos, Dist_Ms : INTEGER;

BEGIN
   Dist_Ms:= Max_Ax1_Ms;
   FOR Pos:= 1 TO 500 DO
      Shape_View[ Pos]:= ROUND(Shape_View[ Pos]* Magnifier/4.0);
   IF (Shape_View[ 0] > 500) THEN
      Shape_View[ 0]:= 500;
   IF (Shape_View[ 1] > 180) THEN
      Shape_View[ 1]:= 180;
   Moveto( Leftaxval, (Bot_Ax1 - Shape_View[ 1]));
   FOR Pos:= 1 TO Shape_View[ 0] DO BEGIN
      IF (Shape_View[ Pos] > 240) THEN
         Shape_View[ Pos]:= 240;
      Lineto( Leftaxval + 3 + Pos, Bot_Ax1 - Shape_View[ Pos]);
   END
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Draw_G2;
  VAR Pos : INTEGER;

  PROCEDURE Sound_Off( Amplitude : INTEGER);

  BEGIN
      Sound(  Amplitude * 250);
      Delay( 5);
  END;

BEGIN          {Of Draw_G2}
   IF (Real_View[ 0] > 500) THEN
       Real_View[ 0]:= 500;
   IF (Real_View[ 1] > 60) THEN
       Real_View[ 1]:= 60;
   Moveto( Leftaxval + 3, Bot_Ax2 - Real_View[ 1]);
   FOR Pos:= 1 TO Real_View[ 0] DO BEGIN
       IF (Real_View[ Pos] > 60) THEN
           Real_View[ Pos]:= 60;
       Lineto( Leftaxval + 3 + Pos, Bot_Ax2 - Real_View[ Pos]);
       Sound_Off( Real_View[ Pos]);
   END;
   Nosound;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


BEGIN                    {Of Draw_Graphs}
   Draw_G1;
   Draw_G2;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


BEGIN                            {Of Draw_Trails}
  IF (Real_View[ 0] <> 0) THEN BEGIN
      Init_Graph;
      Draw_Square( Leftmain, Topmain, Rightmain, Botmain);
      Step_Ms:= ROUND (Max_Ax1_Ms/10.0);
      Draw_Axes;
      Number_Axes( Max_Ax1_Ms);
      Label_Axes;
      Draw_Noise;
      Draw_Graphs( Max_Ax1_Ms);
      IF (NOT Garbage_Flag) AND (Caller = Typing_Call) THEN
          Show_Rise_And_Fall_Positions;
      Write_Name_And_Class;
      End_Graph
  END
  ELSE BEGIN
       Clear_From( 3);
       WRITE('No samples found for trail - press any key to continue...');
       Anykey:= Readkey;
       Clear_From( 3);
  END
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


BEGIN                      {Of Display_Trail Routine}
   FOR I:= 1 TO 550 DO
       IF Disp_Array [I] > 240 THEN  Disp_Array [I] := 240;
   Get_Hour_File_Descrip( Hnam, Systm, Day_Of, Hour_Of, Year_Of);
   Trayl_Num:= Hrec.Trail_Num;
   STR( Trayl_Num, Trail_Nam);
   Trailname:=  'Trail ' + Trail_Nam + ', ' + Systm + ' system, hour ';
   Trailname:= Trailname + Hour_Of + ', day ' + Day_Of + ', ' + Year_Of;
   Draw_Trails (Disp_Array, Trail_Array, Hrec.Background_Noise,
       Trail_Array [0] * 5, Trailname);
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

END.
```

```
UNIT Ts11;
{This unit contains routines which determine features of trails}
{by various numeric and mathematical techniques.                }

Interface

USES Ts1, Dos, Crt, Printer;

PROCEDURE Apply_All_Analysis_Routines_To_Trail;

IMPLEMENTATION

{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Apply_All_Analysis_Routines_To_Trail;
{Applies various statistical and numeric techniques to the  trail}
{in order to find a mathematical description of the trail for use}
{in trail typing.                                                 }

CONST From_Down_Call = TRUE;
       Not_From_Down = FALSE;



{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Write_Numerical_Header;
   VAR Btemp : REAL;
           I : INTEGER;

BEGIN
   Clrscr;
   Normvideo;
   WRITELN('Descriptive features of trail.         Melville/Letschert 1987');
   WRITELN;
   WRITELN('Linear regression : ');
   Lowvideo;
   IF ( B1 < 0.0) THEN BEGIN
      Btemp:= 0.0 - B1;
      WRITELN('   Straight line given by   :   ',B0:8:3, '   - ',Btemp:8:3,'X');
   END
   ELSE WRITELN('   Straight line given by  :   ',B0:8:3, '   + ',B1:8:3,'X');
   WRITE('   This gives variance of ',Variance:8:3);
   WRITELN(' and standard deviation of ',Sqrt(Variance):8:3);
   WRITELN;
   Normvideo;
   WRITE('Amplitude range is from ',Low_Tide,' to ',High_Tide);
   WRITELN(' over duration of ',(Trail_Array[0] * 5),' mSec.');
   WRITELN;
   IF (Top_Total <= 3) THEN
        WRITELN('No upper plateau seems to exist.')
   ELSE
      IF (Top_Total < 10) THEN
           WRITELN('There is a possible upper plateau here.')
      ELSE WRITELN('There are strong signs of an upper plateau.');
   WRITE('Number of fades originally detected was ',Hrec.Number_Of_Fades);
   WRITELN(' whereas program detects ', Fade_Count);
   WRITE('Peak position at ',Peak_Pos * 5,' mSec');
   IF Prob_Plateau THEN
      WRITELN(' - continues on plateau till ',Peak_Pos2 * 5,' mSec.')
   ELSE WRITELN('.');
   IF (Num_Minima > 20) THEN
        Pause_It;
   WRITE('Local minima at ');
   Lowvideo;
   FOR I:= 1 TO (Num_Minima - 1) DO
       WRITE(Minima[I,Xval]*5,':',Minima[I,Yval],', ');
   IF (Num_Minima <> 0) THEN
        WRITELN(Minima[Num_Minima,Xval]*5,':',Minima[Num_Minima,Yval],'.')
   ELSE WRITELN(' - no local minima.');
   Normvideo;
   WRITE('Local maxima at : ');
   Lowvideo;
```

```
    FOR I:= 1 TO (Num_Maxima - 1) DO
        WRITE(Maxima[I,Xval]*5,':',Maxima[I,Yval],', ');
    IF (Num_Maxima <> 0) THEN
         WRITELN(Maxima[Num_Maxima,Xval]*5,':',Maxima[Num_Maxima,Yval],'.')
    ELSE WRITELN(' - no local maxima.');
    WRITELN;
    WRITE('Number extrema : ',Num_Extrema,'   Maxima : ',Num_Maxima);
    WRITELN('  Minima ',Num_Minima);
    Normvideo;
    WRITELN('Linear regression from peak down (on fall) : ');
    Lowvideo;
    IF ( Down_B1 < 0.0) THEN BEGIN
       Btemp:= 0.0 - Down_B1;
      WRITELN('   Straight line given by  :  ',Down_B0:8:3, '   - ',Btemp:8:3,'X');
    END
    ELSE
 ',Down_B1 WRITELN();     Straight  line  given  by    :    ',Down_B0:8:3,  '    +
    WRITE('   This gives variance of ',Down_Variance:8:3);
    WRITELN(' and standard deviation of ',Sqrt(Down_Variance):8:3);
    WRITELN;
    Normvideo;
    WRITELN('Linear regression up to peak (on rise) : ');
    Lowvideo;
    IF ( Up_B1 < 0.0) THEN BEGIN
       Btemp:= 0.0 - Up_B1;
      WRITELN('   Straight line given by  :  ',Up_B0:8:3, '   - ',Btemp:8:3,'X');
    END
   ELSE WRITELN('   Straight line given by  :  ',Up_B0:8:3, '   + ',Up_B1:8:3,'X');
    WRITE('   This gives variance of ',Up_Variance:8:3);
    WRITELN(' and standard deviation of ',Sqrt(Up_Variance):8:3);
    WRITELN('Start down : ',Fallpos,'     End rise : ',Risepos);
    WRITELN('Start down fit = ',Worst_Line_Dif_Pos);
    WRITE('Press any key to continue...');
    Anykey:= Readkey;
    Clrscr
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}


{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Get_Max_And_Min;
  CONST Toll=4; B_Toll=2;
  VAR Heading:(Up,Down);
      X:INTEGER;
      Emax,Emin,Max_Start,Min_Start,Max_Finis,Min_Finis : INTEGER;
  (* *)
  PROCEDURE found_Minimum(Xx,Yy:INTEGER);
  BEGIN;
    IF Num_Minima < 50 THEN
     BEGIN
       Num_Minima:= Num_Minima+1;
       Minima[Num_Minima,Xval] := Xx;
       Minima[Num_Minima,Yval] := Yy - 140
     END
  END;
  (* *)


  PROCEDURE Found_Maximum(Xx,Yy:INTEGER);
  BEGIN;
    IF Num_Maxima < 50 THEN
     BEGIN
       Num_Maxima:= Num_Maxima+1;
       Maxima[Num_Maxima,Xval] := Xx;
       Maxima[Num_Maxima,Yval] := Yy - 140
     END
  END;
  (* *)
```

```
BEGIN
  Num_Maxima:= 0;
  Num_Minima:= 0;
  Emin:= Trail_Array[1];
  Min_Start:=1; Min_Finis:=1;
  Heading:=Down;
  X:=1;
  REPEAT
      X:=X+1;
    IF Heading=Down THEN BEGIN
       IF Trail_Array[x]> (Emin+Toll) THEN
           BEGIN
               Emax:=Trail_Array[x];
               Max_Start:=x;
               Max_Finis:=x;
               Heading:=Up;
               Found_Minimum((Min_Start+Min_Finis) DIV 2,Emin)
           END
         ELSE
             IF Trail_Array[x]>emin THEN {do nothing}
             ELSE
                 IF Trail_Array[x]=Emin THEN Min_Finis:=X
                 ELSE BEGIN
                    Emin:=Trail_Array[x]; Min_Start:=X; Min_Finis:=X
                 END
    END
    ELSE
       {if HEADING=UP then}
        BEGIN
            IF Trail_Array[x]< (Emax-Toll) THEN
                BEGIN
                     Emin:=Trail_Array[x];
                     Min_Start:=x;
                     Min_Finis:=x;
                     Heading:=Down;
                     Found_Maximum((Max_Start+Max_Finis) DIV 2,Emax)
                END
              ELSE
                 IF Trail_Array[x]<emax THEN {do nothing}
                 ELSE
                    IF Trail_Array[x]=Emax THEN Max_Finis:=X
                    ELSE BEGIN
                        Emax:=Trail_Array[x];
                        Max_Start:=X;
                        Max_Finis:=X
                     END
        END
  UNTIL (X >= Trail_Array[0]);
  Found_Minimum( X, Trail_Array[X]);
  Num_Extrema:= Num_Minima + Num_Maxima
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}



{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
FUNCTION Plat_Amp_Average( Plat_In : INTEGER) : REAL;
   VAR Total_Amp : REAL;
     Num_Samp, I : INTEGER;

BEGIN
   Total_Amp:= 0.0;
   FOR I:= Plateaus[ Plat_In, Splatt] TO Plateaus[ Plat_In, Eplatt] DO
       Total_Amp:= Total_Amp + Trail_Array[I];
   Num_Samp:= Plateaus[Plat_In, Eplatt] - Plateaus[Plat_In, Splatt] + 1;
   Plat_Amp_Average:= Total_Amp/Num_Samp
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
```

```
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Get_Plateaus_On_Tolerance;

CONST Plat_Toll = 2;
      Plat_Length_Tol = 3;

VAR          Run_Through, I : INTEGER;
    Start_Platt, End_Platt : INTEGER;
          Startx1, Startx2 : INTEGER;
              Endx1, Endx2 : INTEGER;
                    Inside : BOOLEAN;

BEGIN
  Inside:= FALSE;
  Plat_Count:= 1;
  FOR Run_Through:= 1 TO Trail_Array[0] DO BEGIN
     IF ((Trail_Array[ Run_Through] - 140) >= (High_Tide - Plat_Toll)) THEN
        IF (NOT Inside) THEN BEGIN
              Inside:= TRUE;
              End_Platt:= Run_Through;
              Start_Platt:= Run_Through
         END
         ELSE End_Platt:= Run_Through
     ELSE
        IF Inside THEN BEGIN
         Inside:= FALSE;
         IF ((End_Platt - Start_Platt) > Plat_Length_Tol) THEN BEGIN
             Plateaus[ Plat_Count, Splatt]:= Start_Platt;
             Plateaus[ Plat_Count, Eplatt]:= End_Platt;
             Plat_Count:= Plat_Count + 1
         END
        END;
  END;
  Plat_Count:= Plat_Count - 1;
  Highest_Platt:= 1;
  FOR I:= 2 TO Plat_Count DO
      IF (Plat_Amp_Average( I) > Plat_Amp_Average( Highest_Platt)) THEN
         Highest_Platt:= I;
  IF (Plat_Count > 0) THEN
    Peak_Pos:= Plateaus[ Highest_Platt, Splatt] +
     ((Plateaus[Highest_Platt,Eplatt] - Plateaus[Highest_Platt,Splatt]) DIV 2)
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}




{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Array_Value_Count_And_Plateau_Finder;
    VAR I, Tempint : INTEGER;

BEGIN
    FOR I:= -140 TO -80 DO
        Counts_Array[I]:= 0;
    High_Tide:= -140;
    Low_Tide:= -80;
    FOR I:= 1 TO Trail_Array[0] DO BEGIN
        Tempint:= Trail_Array[I] - 140;
        Counts_Array[ Tempint]:= Counts_Array[ Tempint] + 1;
        IF (Tempint >= High_Tide) THEN BEGIN
            High_Tide:= Tempint;
            Peak_Pos:= I
        END;
        IF (Tempint < Low_Tide) THEN BEGIN
            Low_Tide:= Tempint;
            Min_Pos:= I
        END;
    END;
    Top_Total:= Counts_Array[ High_Tide] + Counts_Array[ High_Tide - 1];
    Prob_Plateau:= (Top_Total > 3);
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
```

```
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Find_Tau ( Inslope : REAL);
BEGIN
   IF (Inslope = 0.0) THEN
      Tau:= 100000000.0
   ELSE Tau:= 1.0 / ((Inslope/5)*1000.0 * Ln(10.0));
   Tau:= 0.0 - Tau;
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}









{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
FUNCTION Variance_From_Line( B_Zero, B_One : REAL; Downer : BOOLEAN;
                                          Posa, Posn : INTEGER) : REAL;
    VAR Expected_Value, Square_Sum : REAL;
        Fall_Dist, Fifteen_Pc_Fall : INTEGER;
                                 I : INTEGER;

BEGIN
   Square_Sum:= 0.0;
   IF Downer THEN BEGIN
      Maxlowdif:= 0.0;
      Worst_Line_Dif_Pos:= Trail_Array[0];
      Fall_Dist:= Trail_Array[0] - Fallpos + 1;
      Fifteen_Pc_Fall:= ROUND (Fall_Dist * 15.0/100.0);
      IF Debug_Flag THEN BEGIN
          Clrscr;
          WRITELN('Differences from expected value.');
      END
   END;
   FOR I:= Posa TO Posn DO BEGIN
       Expected_Value:= B_Zero + (B_One * I);
       IF Downer THEN BEGIN
          IF (((Trail_Array[I] - 140) - Expected_Value) < Maxlowdif) THEN
            IF ((Posa + Fifteen_Pc_Fall) < I) AND
                (I < (Posn - Fifteen_Pc_Fall)) THEN BEGIN
                   Maxlowdif:= (Trail_Array[I] - 140) - Expected_Value;
                   Worst_Line_Dif_Pos:= I;
            END;
          IF Debug_Flag THEN BEGIN
              WRITE('XPOS : ',I,'   DIFRNCE : ');
              WRITELN(Trail_Array[I] - 140 - Expected_Value)
          END
       END;
       Square_Sum:= Square_Sum + Sqr((Trail_Array[I] - 140) - Expected_Value)
   END;
   IF (Downer AND Debug_Flag) THEN
       Anykey:= Readkey;
   IF (Posn <> Posa) THEN
        Variance_From_Line:= Square_Sum/(Posn - Posa)
   ELSE Variance_From_Line:= 0.0
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
```

```
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Linear_Regression_On_Whole_Trail;
   VAR Top_Sum, Low_Sum : REAL;
                      I : INTEGER;

BEGIN
     X_Bar:= (1 + Trail_Array[0])/2;
     Y_Bar:= Hrec.Sample_Mean - 140.0;
     Low_Sum:= 0.0;
     Top_Sum:= 0.0;
     FOR I:= 1 TO Trail_Array[0] DO BEGIN
         Top_Sum:= Top_Sum +
              ((I - X_Bar) * ((Trail_Array[I] - 140) - Y_Bar));
         Low_Sum:= Low_Sum + ((I - X_Bar) * (I - X_Bar));
     END;
     IF (Low_Sum = 0) THEN Low_Sum:= 0.000001;
     B1:= Top_Sum/Low_Sum;
     B0:= Y_Bar - (B1 * X_Bar);
     Variance:= Variance_From_Line( B0, B1, Not_From_Down, 1, Trail_Array[0]);
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}




{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Regression_From_Peak;
   VAR Top_Sum, Low_Sum : REAL;
                      I : INTEGER;

BEGIN
   Top_Sum:= 0.0;
   Low_Sum:= 0.0;
   IF (Plat_Count = 0) THEN BEGIN
        Fallpos:= Trail_Array[0];     {Get last peak value.}
        WHILE ((Trail_Array[Fallpos] - 140) <> High_Tide) DO
              Fallpos:= Fallpos - 1;
   END
   ELSE Fallpos:= Plateaus[Highest_Platt,Eplatt];

   {Now have begin point for regression.}
   Down_X_Bar:= Fallpos + (Trail_Array[0] - Fallpos)/2;
   Down_Y_Bar:= 0.0;
   FOR I:= Fallpos TO Trail_Array[0] DO
       Down_Y_Bar:= Down_Y_Bar + (Trail_Array[I] - 140.0);
   {Getting total Y from peak.}
   Down_Y_Bar:= Down_Y_Bar/(Trail_Array[0] - Fallpos + 1);
   IF (Fallpos <> Trail_Array[0]) THEN BEGIN
     FOR I:= Fallpos TO Trail_Array[0] DO BEGIN
           Top_Sum:= Top_Sum +
               ((I - Down_X_Bar) * ((Trail_Array[I] - 140) - Down_Y_Bar));
           Low_Sum:= Low_Sum + ((I - Down_X_Bar) * (I - Down_X_Bar));
     END;
     IF (Low_Sum = 0) THEN Low_Sum:= 0.000001;
     Down_B1:= Top_Sum/Low_Sum;
     Down_B0:= Down_Y_Bar - (Down_B1 * Down_X_Bar);
   END
   ELSE BEGIN
     Down_B1:= 0.0;
     Down_B0:= Trail_Array[Fallpos]
   END;
   Down_Variance:=
    Variance_From_Line(Down_B0,Down_B1,From_Down_Call,Fallpos,Trail_Array[0]);
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
```

```
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}
PROCEDURE Regression_To_Peak;
   VAR Top_Sum, Low_Sum : REAL;
                Pos1, I : INTEGER;

BEGIN
   Top_Sum:= 0.0;
   Low_Sum:= 0.0;
   Risepos:= 1;      {Get last peak value.}
   WHILE ((Trail_Array[Risepos] - 140) <> High_Tide) DO
         Risepos:= Risepos + 1;
   IF (Plat_Count = 0) OR (Risepos < Plateaus[ Highest_Platt, Splatt]) THEN
        {Do nothing, risepos stays as is.}
   ELSE Risepos:= Plateaus[ Highest_Platt, Splatt];

   {Now have begin point for regression.}
   Up_X_Bar:= ((1 + Risepos))/2;
   Up_Y_Bar:= 0.0;
   FOR I:= 1 TO Risepos DO
       Up_Y_Bar:= Up_Y_Bar + (Trail_Array[I] - 140.0);
   {Getting total Y from peak.}
   Up_Y_Bar:= Up_Y_Bar/Risepos;
   IF (Risepos <> 1) THEN BEGIN
      FOR I:= 1 TO Risepos DO BEGIN
           Top_Sum:= Top_Sum +
                ((I - Up_X_Bar) * ((Trail_Array[I] - 140) - Up_Y_Bar));
           Low_Sum:= Low_Sum + ((I - Up_X_Bar) * (I - Up_X_Bar));
      END;
      IF (Low_Sum = 0) THEN Low_Sum:= 0.000001;
      Up_B1:= Top_Sum/Low_Sum;
      Up_B0:= Up_Y_Bar - (Up_B1 * Up_X_Bar);
   END
   ELSE BEGIN
        Up_B0:= Trail_Array[1] - 140;
        Up_B1:= 0.0;
   END;
   Up_Variance:= Variance_From_Line( Up_B0, Up_B1, Not_From_Down, 1, Risepos);
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}




BEGIN         {All crunch.}
   Get_Max_And_Min;
   Array_Value_Count_And_Plateau_Finder;
   Get_Plateaus_On_Tolerance;
   Linear_Regression_On_Whole_Trail;
   Regression_From_Peak;
   Regression_To_Peak;
   Find_Tau( Down_B1);
   IF (NOT Auto_Type) THEN
      Write_Numerical_Header
END;
{!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!}


END.
```

```
UNIT Ts12;
{This unit contains routines for the interpolation of trail}
{samples for display purposes, as well as routines for  the}
{Fourier  analysis of trails.                              }

Interface

USES Dos, Crt, Printer, Ts1;

PROCEDURE Interpolate_Trail;
PROCEDURE Get_Fourrier_Descriptor;
PROCEDURE Display_Fourrier_Descriptor;


IMPLEMENTATION


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Interpolate_Trail;

VAR  T_Array      : Data_Array;


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Spread_Samples_Over_Range_Of_500;
    VAR Place_To_Put, Num_Points, I : INTEGER;
                      Spread_Factor : REAL;

BEGIN
   Disp_Array[0]:= 500;
   FOR I:= 1 TO 500 DO
       Disp_Array[I]:= 0;
   Num_Points:= T_Array[0];
   IF (Trail_Array[0] <> 1) THEN
        Spread_Factor:= 500.0/(Num_Points - 1)
   ELSE BEGIN
        Spread_Factor:= 500.0;
        Garbage_Flag:= TRUE
   END;
   FOR I:= 2 TO (Num_Points - 1) DO BEGIN
       Place_To_Put:= ROUND( (I - 1) * Spread_Factor);
       Disp_Array[ Place_To_Put]:= T_Array[ I];
   END;
   Disp_Array[1]:= T_Array[1];
   Disp_Array[ 500]:= T_Array[ Num_Points];
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Linear_Interpolate;
   VAR Gapsize, Curpos, I, J : INTEGER;
              Increment_Step : REAL;

BEGIN
  Spread_Samples_Over_Range_Of_500;
  FOR I:= 2 TO 599 DO
    IF (Disp_Array[I] = 0) THEN BEGIN
       Gapsize:= 0;
       Curpos:= I;
       WHILE (Disp_Array[ Curpos] = 0) DO BEGIN
             Curpos:= Curpos + 1;
             Gapsize:= Gapsize + 1
       END;
       Increment_Step:=
           (Disp_Array[Curpos] - Disp_Array[I-1])/(Gapsize + 1);
       {ie. the non-zero values on either side of gap divided by gap size.}
       FOR J:= 1 TO Gapsize DO
           Disp_Array[(I + J - 1)]:=
                 ROUND(Disp_Array[I-1] + (J * Increment_Step))
    END;
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
```

```
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Cubic_Spline_Interpolate;

TYPE  Real_Vector    = ARRAY [0..Cubic_Cutoff] OF REAL;

VAR   Two_D                   : ARRAY [0..Cubic_Cutoff, 1..2] OF INTEGER;
      Cubic_Samples           : INTEGER;
      A_Vector, B_Vector      : Real_Vector;
      C_Vector, D_Vector      : Real_Vector;


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Create_2d_Array;

VAR Index, New_Index       : INTEGER;

BEGIN
  Fillchar (Two_D, Sizeof (Two_D), 0);
  New_Index := -1;
  FOR Index :=  1 TO 500 DO
     IF Disp_Array [Index] <> 0 THEN
      BEGIN
        New_Index:= New_Index + 1;
        Two_D [New_Index, 1] := Index;
        Two_D [New_Index, 2] := Disp_Array [Index]
      END;
  Cubic_Samples := New_Index
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Get_A_B_C_D;

VAR  Z_Vector, U_Vector, L_Vector, Alpha_Vector    : Real_Vector;
      H_Vector : Real_Vector;
     Index, J                                      : INTEGER;

BEGIN
  FOR Index := 0 TO Cubic_Samples - 1 DO
   BEGIN
     A_Vector [Index] := Two_D [Index, 2];
     H_Vector [Index] := Two_D [Index + 1, 1] - Two_D [Index, 1]
   END;
  A_Vector [Cubic_Samples] := Two_D [Cubic_Samples, 2];

  FOR J := 1 TO Cubic_Samples - 1 DO
     Alpha_Vector [J] := (3 * (A_Vector [J + 1] * H_Vector [J - 1] -
        A_Vector [J] * (Two_D [J + 1, 1] - Two_D [J - 1, 1]) +
        A_Vector [J - 1] * H_Vector [J])) /
        (H_Vector [J - 1] * H_Vector [J]);

   L_Vector [0] := 1;
   U_Vector [0] := 0;
   FOR J := 1 TO Cubic_Samples - 1 DO
    BEGIN
      L_Vector [J] := 2 * (Two_D [J + 1, 1] - Two_D [J - 1, 1]) -
         H_Vector [J - 1] * U_Vector [J - 1];
      U_Vector [J] := H_Vector [J] / L_Vector [J]
    END;
    L_Vector [Cubic_Samples] := 1;
    Z_Vector [0] := 0;

    FOR J := 1 TO Cubic_Samples - 1 DO
       Z_Vector [J] := (Alpha_Vector [J] - H_Vector [J - 1] *
          Z_Vector [J - 1]) / L_Vector [J];
    Z_Vector [Cubic_Samples] := 0;

    C_Vector [Cubic_Samples] := 0;
    FOR J := Cubic_Samples - 1 DOWNTO 0 DO
       C_Vector [J] := Z_Vector [J] - U_Vector [J] * C_Vector [J + 1];
```

```
    FOR J := Cubic_Samples - 1 DOWNTO 0 DO
     BEGIN
       B_Vector [J] := (A_Vector [J + 1] - A_Vector [J]) / H_Vector [J] -
          H_Vector [J] * (C_Vector [J + 1] + 2 * C_Vector [J]) / 3;
       D_Vector [J] := (C_Vector [J + 1] - C_Vector [J]) / (3 * H_Vector [J])
     END
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Interpolate_Cubics;

VAR  Current_J, Index        : INTEGER;

BEGIN
  Current_J := 0;
  FOR Index := 1 TO 500 DO
   BEGIN
     IF Index > Two_D [Current_J + 1, 1] THEN  Current_J := Current_J + 1;
     Disp_Array [Index] := ROUND(
        A_Vector [Current_J] +
        B_Vector [Current_J] * (Index - Two_D [Current_J, 1]) +
        C_Vector [Current_J] * (Index - Two_D [Current_J, 1]) *
           (Index - Two_D [Current_J, 1]) +
        D_Vector [Current_J] * (Index - Two_D [Current_J, 1]) *
           (Index - Two_D [Current_J, 1]) * (Index - Two_D [Current_J, 1])
           )
    END
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


BEGIN         {Of Cubic_Spline_Interpolate routine.}
  Spread_Samples_Over_Range_Of_500;
  Create_2d_Array;
  Get_A_B_C_D;
  Interpolate_Cubics
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Compact_Trail;

VAR  Sample_In_1, Sample_In_2, Sample_Out   : INTEGER;
     Sample_Total, Sample_Ave, Index        : INTEGER;
     Sample_Scale                           : REAL;

BEGIN
  Sample_Scale := T_Array [0] / 500.0;
  FOR Sample_Out := 1 TO 500 DO
   BEGIN
     Sample_In_1  := ROUND ((Sample_Out - 1) * Sample_Scale) + 1;
     Sample_In_2  := ROUND (Sample_Out * Sample_Scale);
     Sample_Total  := 0;
     FOR Index := Sample_In_1 TO Sample_In_2 DO
        Sample_Total  := Sample_Total + T_Array [Index];
     Sample_Ave  := ROUND (Float (Sample_Total) /
        (Sample_In_2 - Sample_In_1 + 1));
     Disp_Array [Sample_Out] := Sample_Ave;
   END;
  Disp_Array [0] := 500
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
```

```
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}
PROCEDURE Set_Up_T_Array;

VAR  Sample      : INTEGER;

BEGIN
  FOR Sample := 1 TO Trail_Array [0] DO
      T_Array [Sample] := Trail_Array [Sample] * Db_Scale;
  T_Array [0] := Trail_Array [0]
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


BEGIN                      {Of Interpolate_Trail procedure.}
  Set_Up_T_Array;
  IF T_Array [0] <> 0 THEN
   BEGIN
     IF (T_Array[0] < Cubic_Cutoff) THEN
         Cubic_Spline_Interpolate
     ELSE
       IF (T_Array[0] <= 500) THEN
           Linear_Interpolate
       ELSE Compact_Trail
   END
END;
{^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Get_Fourrier_Descriptor;

VAR  F_Ready       : Data_Array;

PROCEDURE Set_Complex (VAR C1   : Complex; C2    : Complex);

BEGIN
  C1.r := C2.r;
  C1.i := C2.i
END;


PROCEDURE Add_Complex (VAR  C1   : Complex; C2, C3    : Complex);

BEGIN
  C1.r := C2.r + C3.r;
  C1.i := C2.i + C3.i
END;


PROCEDURE Subtract_Complex (VAR  C1   : Complex; C2, C3    : Complex);

BEGIN
  C1.r := C2.r - C3.r;
  C1.i := C2.i - C3.i
END;


PROCEDURE Multiply_Complex (VAR  C1   : Complex; C2, C3    : Complex);

BEGIN
  C1.r := C2.r * C3.r - C2.i * C3.i;
  C1.i := C2.r * C3.i + C2.i * C3.r
END;
```

```
PROCEDURE Divide_Complex (VAR  C1   : Complex; C2, C3    : Complex);

VAR  Divisor      : REAL;

BEGIN
  Divisor := C3.r * C3.r + C3.i * C3.i;
  C1.r := (C2.r * C3.r + C2.i * C3.i) / Divisor;
  C1.i := (C2.i * C3.r - C2.r * C3.i) / Divisor
END;


PROCEDURE Initialise;

VAR  Sample       : INTEGER;

BEGIN
  FOR Sample := 1 TO Trail_Array [0] DO
     Trail_Array [Sample] := Trail_Array [Sample] DIV Db_Scale
END;



PROCEDURE Spread_Samples_Over_Range_Of_512;
    VAR Place_To_Put, Num_Points, I : INTEGER;
                      Spread_Factor : REAL;

BEGIN
   F_Ready[0]:= 512;
   FOR I:= 1 TO 512 DO
       F_Ready[I]:= 0;
   Num_Points:= Trail_Array[0];
   Spread_Factor:= 512.0/(Num_Points - 1);
   FOR I:= 2 TO (Num_Points - 1) DO BEGIN
       Place_To_Put:= ROUND( (I - 1) * Spread_Factor);
       F_Ready[ Place_To_Put]:= Trail_Array[ I];
   END;
   F_Ready[1]:= Trail_Array[1];
   F_Ready[ 512]:= Trail_Array[ Num_Points];
END;



PROCEDURE Linear_Interpolate_Fourrier;
   VAR Gapsize, Curpos, I, J : INTEGER;
               Increment_Step : REAL;

BEGIN
  Spread_Samples_Over_Range_Of_512;
  FOR I:= 2 TO 511 DO
    IF (F_Ready[I] = 0) THEN BEGIN
       Gapsize:= 0;
       Curpos:= I;
       WHILE (F_Ready[ Curpos] = 0) DO BEGIN
             Curpos:= Curpos + 1;
             Gapsize:= Gapsize + 1
       END;
       Increment_Step:=
           (F_Ready[Curpos] - F_Ready[I-1])/(Gapsize + 1);
       {ie. the non-zero values on either side of gap divided by gap size.}
       FOR J:= 1 TO Gapsize DO
           F_Ready[(I + J - 1)]:=
                 ROUND(F_Ready[I-1] + (J * Increment_Step))
    END;
END;
```

```
PROCEDURE Cubic_Spline_Interpolate_Fourrier;

TYPE  Real_Vector    = ARRAY [0..Cubic_Cutoff] OF REAL;

VAR   Two_D                     : ARRAY [0..Cubic_Cutoff, 1..2] OF INTEGER;
      Cubic_Samples             : INTEGER;
      A_Vector, B_Vector        : Real_Vector;
      C_Vector, D_Vector        : Real_Vector;



PROCEDURE Create_2d_Array;

VAR Index, New_Index       : INTEGER;

BEGIN
  Fillchar (Two_D, Sizeof (Two_D), 0);
  New_Index := -1;
  FOR Index :=  1 TO 512 DO
     IF F_Ready [Index] <> 0 THEN
      BEGIN
        New_Index:= New_Index + 1;
        Two_D [New_Index, 1] := Index;
        Two_D [New_Index, 2] := F_Ready [Index]
      END;
  Cubic_Samples := New_Index
END;


PROCEDURE Get_A_B_C_D;

VAR   Z_Vector, U_Vector, L_Vector, Alpha_Vector    : Real_Vector;
       H_Vector : Real_Vector;
      Index, J                                      : INTEGER;

BEGIN
  FOR Index := 0 TO Cubic_Samples - 1 DO
   BEGIN
     A_Vector [Index] := Two_D [Index, 2];
     H_Vector [Index] := Two_D [Index + 1, 1] - Two_D [Index, 1]
   END;
  A_Vector [Cubic_Samples] := Two_D [Cubic_Samples, 2];

  FOR J := 1 TO Cubic_Samples - 1 DO
     Alpha_Vector [J] := (3 * (A_Vector [J + 1] * H_Vector [J - 1] -
        A_Vector [J] * (Two_D [J + 1, 1] - Two_D [J - 1, 1]) +
        A_Vector [J - 1] * H_Vector [J])) /
        (H_Vector [J - 1] * H_Vector [J]);

   L_Vector [0] := 1;
   U_Vector [0] := 0;
   FOR J := 1 TO Cubic_Samples - 1 DO
    BEGIN
      L_Vector [J] := 2 * (Two_D [J + 1, 1] - Two_D [J - 1, 1]) -
         H_Vector [J - 1] * U_Vector [J - 1];
      U_Vector [J] := H_Vector [J] / L_Vector [J]
    END;
    L_Vector [Cubic_Samples] := 1;
    Z_Vector [0] := 0;

    FOR J := 1 TO Cubic_Samples - 1 DO
       Z_Vector [J] := (Alpha_Vector [J] - H_Vector [J - 1] *
          Z_Vector [J - 1]) / L_Vector [J];
    Z_Vector [Cubic_Samples] := 0;

    C_Vector [Cubic_Samples] := 0;
    FOR J := Cubic_Samples - 1 DOWNTO 0 DO
       C_Vector [J] := Z_Vector [J] - U_Vector [J] * C_Vector [J + 1];
```

```
    FOR J := Cubic_Samples - 1 DOWNTO 0 DO
     BEGIN
       B_Vector [J] := (A_Vector [J + 1] - A_Vector [J]) / H_Vector [J] -
          H_Vector [J] * (C_Vector [J + 1] + 2 * C_Vector [J]) / 3;
       D_Vector [J] := (C_Vector [J + 1] - C_Vector [J]) / (3 * H_Vector [J])
     END
END;


PROCEDURE Interpolate_Cubics;

VAR  Current_J, Index        : INTEGER;

BEGIN
  Current_J := 0;
  FOR Index := 1 TO 512 DO
   BEGIN
     IF Index > Two_D [Current_J + 1, 1] THEN  Current_J := Current_J + 1;
     F_Ready [Index] := ROUND(
        A_Vector [Current_J] +
        B_Vector [Current_J] * (Index - Two_D [Current_J, 1]) +
        C_Vector [Current_J] * (Index - Two_D [Current_J, 1]) *
           (Index - Two_D [Current_J, 1]) +
        D_Vector [Current_J] * (Index - Two_D [Current_J, 1]) *
           (Index - Two_D [Current_J, 1]) * (Index - Two_D [Current_J, 1])
           )
    END
END;


BEGIN
  Spread_Samples_Over_Range_Of_512;
  Create_2d_Array;
  Get_A_B_C_D;
  Interpolate_Cubics
END;




PROCEDURE Compact_Trail_Fourrier;

VAR  Sample_In_1, Sample_In_2, Sample_Out        : INTEGER;
     Sample_Total, Sample_Ave, Index             : INTEGER;
     Sample_Scale                                : REAL;


BEGIN
  Sample_Scale := Trail_Array [0] / 512.0;
  FOR Sample_Out := 1 TO 512 DO
   BEGIN
     Sample_In_1 := ROUND ((Sample_Out - 1) * Sample_Scale) + 1;
     Sample_In_2 := ROUND (Sample_Out * Sample_Scale);
     Sample_Total := 0;
     FOR Index := Sample_In_1 TO Sample_In_2 DO
        Sample_Total := Sample_Total + Trail_Array [Index];
     Sample_Ave := ROUND (Float (Sample_Total) /
        (Sample_In_2 - Sample_In_1 + 1));
     F_Ready [Sample_Out] := Sample_Ave;
   END;
  F_Ready [0] := 512
END;
```

```
PROCEDURE Interpolate_Trail_Fourrier;

BEGIN
  IF (Trail_Array[0] < Cubic_Cutoff) THEN
       Cubic_Spline_Interpolate_Fourrier
  ELSE
    IF (Trail_Array[0] <= 512) THEN
         Linear_Interpolate_Fourrier
    ELSE Compact_Trail_Fourrier
END;



PROCEDURE Ready_Fourrier_Array (VAR  Power     : INTEGER);

VAR  Sample      : INTEGER;

BEGIN
  Interpolate_Trail_Fourrier;
  Power := 9;
  FOR Sample := 1 TO 512 DO
   BEGIN
     Fourrier^ [Sample].R := Float (Sample);
     Fourrier^ [Sample].I := Float (F_Ready [Sample])
   END
 END;


PROCEDURE Ready_Fourrier_Array (VAR  Max_Power  : INTEGER);

TYPE Comp_Bound_Struct = ARRAY [1..20120] OF INTEGER;
     Comp_Bound        = ^comp_Bound_Struct;

VAR  Boundry_Points                 : INTEGER;
     Point, Index, Axis, Sample     : INTEGER;
     Scale                          : REAL;
     Imaginary_Boundry              : Comp_Bound;
     Real_Boundry                   : Comp_Bound;

BEGIN

  {-------------------------------------------------------------}
  {-  Circle boundry clockwise getting complex boundry points. -}
  {-------------------------------------------------------------}

  NEW (Real_Boundry);
  NEW (Imaginary_Boundry);
  Index := 1;
  FOR Sample := 1 TO Trail_Array [0] DO
   BEGIN
     Real_Boundry^ [Index] := Sample;
     Imaginary_Boundry^ [Index] := Trail_Array [Sample];
     Index := SUCC (Index)
   END;
  FOR Axis := Trail_Array [Trail_Array [0]] DOWNTO 0 DO
   BEGIN
     Real_Boundry^ [Index] := Trail_Array [0];
     Imaginary_Boundry^ [Index] := Axis;
     Index := SUCC (Index)
   END;
  FOR Axis := Trail_Array [0] DOWNTO 0 DO
   BEGIN
     Real_Boundry^ [Index] := Axis;
     Imaginary_Boundry^ [Index] := 0;
     Index := SUCC (Index)
   END;
  FOR Axis := 0 TO Trail_Array [1] DO
   BEGIN
     Real_Boundry^ [Index] := 0;
     Imaginary_Boundry^ [Index] := Axis;
     Index := SUCC (Index)
   END;
```

```
  {-------------------------------------------------------}
  {-  Put boundry points into Fourrier array anticlockwise.  -}
  {-  Note number of points compacted to closest smaller     -}
  {-  power of 2. Max power of 2 is 11, ie. 2048 points.     -}
  {-------------------------------------------------------}

  Boundry_Points := 2 * Trail_Array [0] + Trail_Array [1] +
     Trail_Array [Trail_Array [0]];
  Max_Power := 1;
  WHILE (Max_Power <= 11) AND (1 shl Max_Power <= Boundry_Points) DO
     Max_Power := Max_Power + 1;
  Max_Power := Max_Power - 1;
  Scale := Float (Boundry_Points) / Float (1 shl Max_Power);
  Index := 1;
  Point := Boundry_Points;
  WHILE Point >= 1 DO
   BEGIN
     Fourrier^ [Index].r := Real_Boundry^ [Point];
     Fourrier^ [Index].i := Imaginary_Boundry^ [Point];
     Point := Boundry_Points - ROUND (Index * Scale);
     Index := SUCC (Index)
   END;
  DISPOSE (Real_Boundry);
  DISPOSE (Imaginary_Boundry)
END;


*)
 (*

PROCEDURE Ready_Fourrier_Array (VAR  Max_Power  : INTEGER);


VAR  Point, Index            : INTEGER;
     Scale                   : REAL;

BEGIN

  {-------------------------------------------------------}
  {-  Put boundry points into Fourrier array anticlockwise.  -}
  {-  Note number of points compacted to closest smaller     -}
  {-  power of 2. Max power of 2 is 11, ie. 2048 points.     -}
  {-------------------------------------------------------}

  Max_Power := 1;
  WHILE (Max_Power <= 11) AND (1 shl Max_Power <= Trail_Array [0]) DO
     Max_Power := Max_Power + 1;
  Max_Power := Max_Power - 1;
  Scale := Float (Trail_Array [0]) / Float (1 shl Max_Power);
  Index := 1;
  Point := 1;
  WHILE Point <= Trail_Array [0] DO
   BEGIN
     Fourrier^ [Index].r := Float (Point);
     Fourrier^ [Index].i := Float (Trail_Array [Point]);
     Point := ROUND (Index * Scale) + 1;
     Index := SUCC (Index)
   END
END;
*)
```

```
PROCEDURE Perform_Fft (Ln    : INTEGER);

CONST  Pi = 3.141592654;

VAR  U, W, T, Temp                              : Complex;
     N, Nm1, Nv2, J, I, K, L, Le, Le1, Ip       : INTEGER;


BEGIN
  N := 1 shl Ln;
  Nv2 := N DIV 2;
  Nm1 := N - 1;
  J := 1;

  FOR I := 1 TO Nm1 DO
   BEGIN
     IF I < J THEN
      BEGIN
        Set_Complex (T, Fourrier^ [J]);
        Set_Complex (Fourrier^ [J], Fourrier^ [I]);
        Set_Complex (Fourrier^ [I], T)
      END;
     K := Nv2;
     WHILE K < J DO
      BEGIN
        J := J - K;
        K := K DIV 2
      END;
     J := J + K
   END;

  FOR L := 1 TO Ln DO
   BEGIN
     Le := 1 shl L;
     Le1 := Le DIV 2;
     U.r := 1.0;
     U.i := 0.0;
     W.r := Cos (Pi / Le1);
     W.i := Sin (Pi / Le1) * -1;
     FOR J := 1 TO Le1 DO
      BEGIN
        I := J;
        WHILE I <= N DO
         BEGIN
           Ip := I + Le1;
           Multiply_Complex (T, Fourrier^ [Ip], U);
           Subtract_Complex (Fourrier^ [Ip], Fourrier^ [I], T);
           Add_Complex (Fourrier^ [I], Fourrier^ [I], T);
           I := I + Le;
         END;
        Multiply_Complex (U, U, W)
      END
   END;
  FOR I := 1 TO N DO
   BEGIN
     Temp.r := Float (N);
     Temp.i := 0.0;
     Divide_Complex (Fourrier^ [I], Fourrier^ [I], Temp)
   END
END;
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Normalise_Fourrier_Identifier;

VAR  Divisor                    : Complex;
     Sample, Stop_Point         : INTEGER;

BEGIN
  Set_Complex (Divisor, Fourrier^ [1]);
  IF Fourrier_Points > 24 THEN  Stop_Point := 24
  ELSE                          Stop_Point := Fourrier_Points;
  FOR Sample := 1 TO Stop_Point DO
    Divide_Complex (Fourrier^ [Sample], Fourrier^ [Sample], Divisor)
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


BEGIN
  Ready_Fourrier_Array (Power);
  Perform_Fft (Power);
  Fourrier_Points := 1 shl Power;
  Normalise_Fourrier_Identifier
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}



{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Display_Fourrier_Descriptor;

VAR  Sample      : INTEGER;
     Junk        : CHAR;
     Stop_Point  : INTEGER;

BEGIN
  Clrscr;
  IF Fourrier_Points > 24 THEN  Stop_Point := 24
  ELSE                          Stop_Point := Fourrier_Points;
  FOR Sample := 1 TO Stop_Point DO
     WRITELN (Fourrier^ [Sample].r :20:9, Fourrier^ [Sample].i :20:9);
  Junk:= Readkey
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


END.
```

```
UNIT Ts13;
{This unit contains driver routines and utilities.}

Interface

USES Dos, Crt, Printer, Ts1, Ts2, Ts3, Ts4, Ts5,
     Ts6, Ts7, Ts8, Ts9, Ts10, Ts11, Ts12;


PROCEDURE Get_Next_Trail_Desired;
PROCEDURE Next_Choice;
FUNCTION Check_Conditions( Rule_In : Rules; VAR Badcond : BYTE) : BOOLEAN;
PROCEDURE Find_A_Triggered_Rule( VAR Rule_Back : Rules;
                                 VAR Rule_Num : INTEGER);
PROCEDURE Drive_Rules( VAR Type_Chosen : BYTE; VAR Rule_Back : Rules);
PROCEDURE Tell_Typing( Rule_Used : Rules; First_Fired : INTEGER);
PROCEDURE High_Up_Write( Str_In : Str80);
PROCEDURE Show_Options_Open;
PROCEDURE Handle_Option;
PROCEDURE Type_Trail;
PROCEDURE Auto_Type_File (Header_File : Str12; Sel_Choice : CHAR;
                          Daystr : String);



IMPLEMENTATION


{*********************************************************************}
PROCEDURE Get_Next_Trail_Desired;
    VAR Choice_Char : CHAR;

BEGIN
  Gotoxy (1,24);
  WRITE ('Hit "N" for next trail else "S" to specify trail.');
  Choice_Char := Get_Valid_Choice_From ([Esc, 'N', 'S']);
  CASE Choice_Char OF
       Esc : Exitting:= TRUE;
       'S' : Get_Trail;
       'N' : Get_Next_Trail;
  END;
END;
{*********************************************************************}



{???????????????????????????????????????????????????????????????????}
PROCEDURE Next_Choice;
{Determines what the next rule to have been fired  would have been}
{if the one that was fired did not exist, and shows  the user  the}
{typing that would have been made as well as the rule. If no other}
{rule  could  have  been  fired  then a message to  that effect is}
{printed. Note that NEXT_CHOICE will reset of  BEFORE_NEXT_CHOICE,}
{so the user can ask for the next next choice, etcetera.          }



{???????????????????????????????????????????????????????????????????}
PROCEDURE Get_Pos_In_File;
   VAR R_Num : INTEGER;

BEGIN
   SEEK( Pribase, 0);
   REPEAT
        READ( Pribase, R_Num)
   UNTIL (R_Num = Before_Next_Choice)
END;
{???????????????????????????????????????????????????????????????????}
```

```
{??????????????????????????????????????????????????????????????????????}
FUNCTION Check_Out_Rule( Rule_In : Rules; VAR Badcond : BYTE) : BOOLEAN;
{Tests to see if all conditions in a given rule hold.}
   VAR Cond_Check_Cnt, Num_Conds : BYTE;
                       Still_Good : BOOLEAN;
                         Cur_Cond : Condition;

BEGIN
   Cond_Check_Cnt:= 1;
   Still_Good:= TRUE;
   Num_Conds:= Rule_In[0].Cond_Num;
   WHILE (Still_Good AND (Cond_Check_Cnt <= Num_Conds)) DO BEGIN
         Cur_Cond:= Rule_In[ Cond_Check_Cnt];
         Still_Good:= Condition_Checker( Cur_Cond);
         Cond_Check_Cnt:= Cond_Check_Cnt + 1
   END;
   IF Still_Good THEN
        Check_Out_Rule:= TRUE       {Rule triggered.}
   ELSE BEGIN
        Badcond:= Cond_Check_Cnt - 1;
        Check_Out_Rule:= FALSE
    END;
END;
{??????????????????????????????????????????????????????????????????????}



{??????????????????????????????????????????????????????????????????????}
PROCEDURE Drive_Until_Trigger;
{Repeatedly reads priority file entries and reads the rules}
{which correspond to these entries until rule  found  where}
{all conditions satisfied. If no such rule found then  sets}
{BEFORE_NEXT_CHOICE to 0 as signal, else  sets  it  to  the}
{number of the rule triggered.                              }
   VAR Rule_Back : Rules;
        Badcond : BYTE;
          Found : BOOLEAN;
          R_Num : INTEGER;

BEGIN
   Found:= FALSE;
   Fillchar( Failures, Sizeof( Failures), 0);
   Test_Failures:= 0;                    {Initialising 'Why not' variables.}
   WHILE ((NOT(EOF( Pribase))) AND (NOT Found)) DO BEGIN
        READ( Pribase, R_Num);
        SEEK( Rbase, (R_Num - 1));
        READ( Rbase, Rule_Back);
        Found:= Check_Out_Rule( Rule_Back, Badcond);
        IF (NOT Found) THEN BEGIN
            Test_Failures:= Test_Failures + 1;
            Failures[ Test_Failures].Rnumber:= R_Num;
            Failures[ Test_Failures].Cnumber:= Badcond;
        END;
   END;
   IF Found THEN BEGIN
        Last_Fired:= Rule_Back;
        Before_Next_Choice:= R_Num
   END
   ELSE Before_Next_Choice:= 0
END;
{??????????????????????????????????????????????????????????????????????}
```

```
{??????????????????????????????????????????????????????????????????????????}
PROCEDURE Find_Action_And_Write_Rule;
{Determines the trail typing that would have been given from the}
{action part of LAST_FIRED, and writes the rule involved.       }
    VAR Trail_Would_Choose, Num_Conds, I : BYTE;

BEGIN
   Determine_Action( Last_Fired[20], Trail_Would_Choose);
   WRITELN('Next choice would have been :');
   Show_Act( Last_Fired[20]);
   WRITE('This would follow from firing of rule ');
   WRITELN( Before_Next_Choice);
   WRITELN('This rule is fired given the following conditions :');
   Num_Conds:= Last_Fired[0].Cond_Num;
   FOR I:= 1 TO Num_Conds DO
        Show_Cond( Last_Fired[I]);
   WRITELN;
   WRITE('Want to see the rules which failed (Y/N)? ');
   IF (Upcase( Readkey) = 'Y') THEN
         Show_Why_Nots
END;
{??????????????????????????????????????????????????????????????????????????}




BEGIN                                   {Of NEXT_CHOICE  procedure.}
   Clear_From( 8);
   Normvideo;
   WRITELN('         NEXT CHOICE DECISION :');
   Lowvideo;
   IF (Before_Next_Choice = 0) THEN
        WRITELN('No other rules triggered => no other choices.')
   ELSE BEGIN
        Get_Pos_In_File;
        Drive_Until_Trigger;
        IF (Before_Next_Choice = 0) THEN
             WRITELN('No other rules triggered => no other choices.')
        ELSE Find_Action_And_Write_Rule
   END;
END;
{??????????????????????????????????????????????????????????????????????????}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
FUNCTION Check_Conditions( Rule_In : Rules; VAR Badcond : BYTE) : BOOLEAN;
{Calls CONDITION_CHECKER to test each condition in the rule}
{passed to it until either a condition found false  or  all}
{the conditions tested. Function set false if condition not}
{satisfied, set true if all conditions okay.               }
   VAR Cond_Check_Cnt, Num_Conds : BYTE;
                      Still_Good : BOOLEAN;
                        Cur_Cond : Condition;


BEGIN
   Cond_Check_Cnt:= 1;
   Still_Good:= TRUE;
   Num_Conds:= Rule_In[0].Cond_Num;
   WHILE (Still_Good AND (Cond_Check_Cnt <= Num_Conds)) DO BEGIN
         Cur_Cond:= Rule_In[ Cond_Check_Cnt];
         Still_Good:= Condition_Checker( Cur_Cond);
         (* *)
         IF (NOT Still_Good) AND Debug_Flag THEN BEGIN
             WRITELN(' died on condition ',Cur_Cond.Cond_Num);
             Anykey:= Readkey
         END;
         (* *)
         Cond_Check_Cnt:= Cond_Check_Cnt + 1
   END;
   IF Still_Good THEN
         Check_Conditions:= TRUE        {Rule triggered.}
   ELSE BEGIN
         Badcond:= Cond_Check_Cnt - 1;     {Was incremented before loop exit.}
         Check_Conditions:= FALSE;
   END;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Find_A_Triggered_Rule( VAR Rule_Back : Rules;
                                 VAR Rule_Num : INTEGER);
{Repeatedly reads integers from the priority file specified in the}
{parameter RELEVANT,  and gets  the  rules corresponding to  these}
{integers. Does this until either determines that a rule triggered}
{(all conditions satisfied), or end of priority file reached.     }
     VAR Badcond : BYTE;
           Found : BOOLEAN;

BEGIN
   Fillchar( Failures, Sizeof( Failures), 0);
   Test_Failures:= 0;                    {Initialising 'Why not' variables.}
   Found:= FALSE;
   RESET( Pribase);
   WHILE ((NOT (EOF( Pribase))) AND (NOT Found)) DO BEGIN
        READ( Pribase, Rule_Num);
        IF Debug_Flag THEN
           WRITE('Checking rule ',Rule_Num, ' : ');
        SEEK( Rbase, (Rule_Num - 1));   {Rule nums 1 on, file starts at 0.}
        READ( Rbase, Rule_Back);
        Found:= Check_Conditions( Rule_Back, Badcond);
        IF (NOT Found) THEN BEGIN      {Get the 'Why Not?' information.   }
             Test_Failures:= Test_Failures + 1;
             Failures[ Test_Failures].Rnumber:= Rule_Num;
             Failures[ Test_Failures].Cnumber:= Badcond;
        END;
   END;

   IF (NOT Found) THEN Rule_Num:= 0
                                      {Signal that no rule triggered.}
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Drive_Rules( VAR Type_Chosen : BYTE; VAR Rule_Back : Rules);
{Determines which priority file to use, according to whether opening}
{lead, later lead, or following to trick. Calls for a triggered rule}
{from this file to be found, and determines the card to play  from}
{the action part of the rule if a rule found. If not then a  message}
{is sent to the user to warn of the fatal flaw in the rule base.    }
    VAR Rule_Num : INTEGER;
        Pri_File : BYTE;

BEGIN

  Find_A_Triggered_Rule( Rule_Back, Rule_Num);

  IF (Rule_Num = 0) THEN
       Send_Crash_Message_And_Abort
  ELSE BEGIN
       Determine_Action( Rule_Back[20], Type_Chosen);
       Last_Fired:= Rule_Back;        {Data for query/next-choice.}
       Before_Next_Choice:= Rule_Num
  END
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}




{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Tell_Typing( Rule_Used : Rules; First_Fired : INTEGER);
   VAR Num_Conds, I : INTEGER;

BEGIN
   Gotoxy( 1,4);
   Normvideo;
   WRITE('Trail ', Hrec.Trail_Num,'.');
   WRITE(' The following typing was made, in accordance with Rule ');
   WRITELN( First_Fired,' :');
   WRITE('   ');
   Show_Act( Rule_Used[20]);
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}




{************************************************************************}
PROCEDURE High_Up_Write( Str_In : Str80);
  VAR I : INTEGER;

BEGIN
  FOR I:= 1 TO LENGTH(Str_In) DO BEGIN
      IF (Str_In[I] = Upcase( Str_In[I])) THEN
          Normvideo
      ELSE Lowvideo;
      WRITE(Str_In[I]);
  END
END;
{************************************************************************}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Show_Options_Open;

BEGIN
   Gotoxy(1,1);
   Normvideo;
   WRITE('TrailStar  Typing  System.                        ');
   WRITELN('    Melville/Letschert (1987)');;
   Gotoxy(1,2);
   WRITE('Options : ');
   Lowvideo;
   High_Up_Write('H = Help. W = Why. N = Next choice. ');
   High_Up_Write('I = Instruct. T = Typing. X = eXit');
   WRITELN;
   Gotoxy(1,24);
   WRITE('Please enter an option : ');
   Play_Choice:= Get_Valid_Choice_From(['H','W','N','I','T','X'])
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}



{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Handle_Option;
{Passes control to the routine appropriate to the particular}
{option chosen by the user, and then restores  screen  after}
{that routine ends.                                         }
BEGIN
   CASE Play_Choice OF
        'H' : Do_Help;
        'I' : Learn;
        'N' : Next_Choice;
        'T' : Play_Choice:= 'T';    {Type another trail.}
        'W' : Query;
        'X' : Play_Choice:= 'X'
   END;
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Type_Trail;
{Main driver for user trail typing.}
    VAR Type_Chosen : BYTE;
          Rule_Back : Rules;

BEGIN
   Play_Choice:= 'T';
   REPEAT
     IF (Play_Choice = 'T') THEN BEGIN
       Get_Next_Trail_Desired;
       IF (NOT Exitting) THEN BEGIN
         Gotoxy( 1, 24);
         Clreol;
         Gotoxy(1,24);
         WRITE('Analysing the trail...');
         IF (Hrec.Num_Samples < 2) THEN BEGIN
           Clear_From( 3);
           WRITELN('Too few samples (',Hrec.Num_Samples,') - press any key.');
           Anykey:= Readkey;
           Clear_From( 3);
           Garbage_Flag:= TRUE
         END
         ELSE BEGIN
           IF (NOT Garbage_Flag) THEN
               Apply_All_Analysis_Routines_To_Trail;
           Gotoxy( 1, 24);
           WRITE('Interpolating trail...');
           Interpolate_Trail;
           Display_Trail( Typing_Call);
           Drive_Rules( Type_Chosen, Rule_Back);
           First_Choice:= Before_Next_Choice;
           First_Fired:= Rule_Back;
         END
       END;
     END;
     REPEAT
        IF (NOT Exitting) THEN BEGIN
             Tell_Typing( Rule_Back, First_Choice);
             Show_Options_Open;
             Clear_From( 5);
             Handle_Option;
        END
        ELSE BEGIN
             Exitting:= FALSE;
             Clrscr;
             Gotoxy( 1,4);
             WRITELN('No typing made as exit requested.');
             WRITE('Press any key to return to main menu...');
             Pause;
             Exit
        END;
     UNTIL (Play_Choice IN ['T','X']);
  UNTIL (Play_Choice = 'X');
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
PROCEDURE Auto_Type_File (Header_File : Str12; Sel_Choice : CHAR;
                          Daystr : String);

VAR  Type_Chosen                : BYTE;
     Rule_Back                  : Rules;
     Sample_File                : Str12;
     Header_Rec                 : Trail_Header;
     Start_Trail, Stop_Trail    : INTEGER;
     Start_Pos                  : INTEGER;
     Answer                     : Str8;
     Start_Trail_Str            : Str12;
     Stop_Trail_Str             : Str12;
     Sillykey                   : Str8;


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
FUNCTION No_Disqualification : BOOLEAN;
   VAR Same : BOOLEAN;
          I : INTEGER;

BEGIN
   IF (Sel_Choice = 'D') THEN BEGIN
       Same:= TRUE;
       FOR I:= 1 TO LENGTH( Daystr) DO
           IF (Header_File[ I] <> Daystr[ I]) THEN
               Same:= FALSE;
       No_Disqualification:= Same
   END
   ELSE No_Disqualification:= TRUE
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}


{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
FUNCTION set_up_auto_type_ok : BOOLEAN;
  VAR Old_Dir : String;

BEGIN
   Menu_Head;
   Getdir( 0, Old_Dir);
   Chdir( Path);
   Sample_File := Header_File;
   Sample_File [1] := 'S';
   ASSIGN (Trail_Head, Header_File);
   RESET (Trail_Head);
   ASSIGN (Trail_Sample, Sample_File);
   {$I-}  RESET (Trail_Sample);  {$I+}
   set_up_auto_type_ok:= (ioresult = 0);
   Chdir( Old_Dir);
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
BEGIN
  IF No_Disqualification THEN BEGIN
     IF (NOT set_up_Auto_type_ok) THEN
        Sillykey:= Users_Response_To (1, 'File ' + Sample_File +
            ' not on current drive (OK)?')
     ELSE BEGIN
      READ (Trail_Head, Header_Rec);
      Start_Trail := Header_Rec.Trail_Num;
      SEEK (Trail_Head, Filesize (Trail_Head) - 1);
      READ (Trail_Head, Header_Rec);
      Stop_Trail := Header_Rec.Trail_Num;
      SEEK(Trail_Head,0);
      IF First_File_To_Auto_Type THEN BEGIN
         First_File_To_Auto_Type := FALSE;
         REPEAT
           Answer := Users_Response_To (1,
              'Start at beginning of file (Y/N)?');
         UNTIL ((Answer = 'Y') OR (Answer = 'N'));
         IF Answer = 'N' THEN BEGIN
            STR (Start_Trail, Start_Trail_Str);
            STR (Stop_Trail, Stop_Trail_Str);
            Start_Pos := Get_User_Int ('Enter trail between ' +
               Start_Trail_Str + ' and ' + Stop_Trail_Str + ' :  ',
               Start_Trail, Stop_Trail);
            Header_Rec.Trail_Num := -1;
            WHILE (NOT (EOF (Trail_Head))) AND
                  (Header_Rec.Trail_Num <> Start_Pos) DO
               READ (Trail_Head, Header_Rec);
            SEEK (Trail_Head, Filepos (Trail_Head) - 1);
         END
      END;
      Gotoxy (5, 5);
      WRITE ('Working on file ', Header_File, ' with trail range :  ');
      WRITE (Start_Trail, '  -  ', Stop_Trail, '.');
      Gotoxy (5, 7);
      WRITE ('Currently processing trail  ');
      WHILE NOT (EOF (Trail_Head)) DO BEGIN
         IF Keypressed THEN
            IF (Readkey = Esc) THEN BEGIN
                 CLOSE( Trail_Sample);
                 CLOSE( Trail_Head);
                 Exitting:= TRUE;
                 Exit
             END;
         Get_Next_Trail;
         Gotoxy (34, 7);
         WRITE (Hrec.Trail_Num);
         IF ((Sel_Choice = 'T') AND (Hrec.Trail_Type <> 255)) THEN
             WRITE(' - pre-typed')
         ELSE BEGIN
             IF ((Hrec.Num_Samples < 2) OR (Hrec.Elapsed_Time = 0.0)) THEN
                Garbage_Flag:= TRUE;
             IF NOT Garbage_Flag THEN
                Apply_All_Analysis_Routines_To_Trail;
             Drive_Rules (Type_Chosen, Rule_Back);
             First_Choice := Before_Next_Choice;
             First_Fired := Rule_Back
          END
      END;
      CLOSE (Trail_Sample)
    END;
    CLOSE (Trail_Head)
  END
END;
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

END.
```

Appendix I - Source Listing of the Library Unit

(Used by all the programs listed in Appendices J to O)

```
UNIT Library;
{Contains various library routines used in programs written}
{by Stuart Melville. This version is that of November, 1991}

Interface

USES Dos, Crt, Printer, Graph;


TYPE Charset = SET OF CHAR;
       Str80 = String[80];

VAR Graphdriver, Graphmode, Errorcode : INTEGER;
                                 Regis : Registers;

PROCEDURE Menu_Head( Strin : Str80);
PROCEDURE Write_Centre( Strin : Str80);
FUNCTION Get_Valid_Choice_From( Valchars : Charset) : CHAR;
FUNCTION Echo_Valid_Choice_From( Valchars : Charset) : CHAR;
FUNCTION Get_Good_Int( Lowb, Upb : INTEGER) : INTEGER;
PROCEDURE Highlight( Massage : String);
PROCEDURE Write_At( Col, Row : BYTE; Message : String);
PROCEDURE Draw_Box( Row1, Row2, Col1, Col2 : BYTE);
PROCEDURE Init_Graph;
PROCEDURE End_Graph;
PROCEDURE Hardcopy( Inverse : BOOLEAN; Mode : BYTE);
PROCEDURE Curson;
PROCEDURE Cursoff;
PROCEDURE Inverse;
PROCEDURE Normal;
PROCEDURE Pause;
PROCEDURE Alarm;
PROCEDURE Clear_From( Screen_Line : BYTE);
PROCEDURE Drawlogo;

IMPLEMENTATION


{-------------------------------------------------------------------}
PROCEDURE Inverse;

BEGIN
   Textbackground(7);
   Textcolor(0);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Normal;

BEGIN
   Textbackground(0);
   Textcolor(7);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Menu_Head( Strin : Str80);
   VAR I, Offset : BYTE;

BEGIN
   Clrscr;
   Offset:= (80 - LENGTH( Strin)) DIV 2;
   FOR I:= 1 TO Offset DO
       WRITE(' ');
   Inverse;
   WRITELN( Strin);
   Normal;
   Gotoxy( 1,5);
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Write_Centre( Strin : Str80);
   VAR I, Offset : BYTE;

BEGIN
   Offset:= (80 - LENGTH( Strin)) DIV 2;
   FOR I:= 1 TO Offset DO
       WRITE(' ');
   WRITELN( Strin);
END;
{-------------------------------------------------------------------------}



{-------------------------------------------------------------------------}
FUNCTION Get_Valid_Choice_From( Valchars : Charset) : CHAR;
  VAR Tmp : CHAR;

BEGIN
   REPEAT
      Tmp:= Readkey;
      Tmp:= Upcase(Tmp);
   UNTIL (Tmp IN Valchars);
   Get_Valid_Choice_From:= Tmp;
END;
{-------------------------------------------------------------------------}



{-------------------------------------------------------------------------}
FUNCTION Echo_Valid_Choice_From( Valchars : Charset) : CHAR;
  VAR Tmp : CHAR;

BEGIN
   REPEAT
      Tmp:= Readkey;
      Tmp:= Upcase(Tmp);
   UNTIL (Tmp IN Valchars);
   WRITE( Tmp);
   Echo_Valid_Choice_From:= Tmp;
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
FUNCTION Get_Good_Int( Lowb, Upb : INTEGER) : INTEGER;
   VAR  Ival, Err : INTEGER;
           Intstr : Str80;

BEGIN
   REPEAT
        READLN( Intstr);
        VAL( Intstr, Ival, Err);
        IF ((Err <> 0) OR (Ival < Lowb) OR (Ival > Upb)) THEN BEGIN
             Err:= 1;
             WRITE('Invalid entry - want integer in range ',Lowb,'..',Upb);
             WRITE('  - please retry : ');
        END;
   UNTIL (Err = 0);
   Get_Good_Int:= Ival
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Highlight( Massage : String);

BEGIN
   Inverse;
   WRITE( Massage);
   Normal;
END;
{-------------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Write_At( Col, Row : BYTE; Message : String);

BEGIN
   Gotoxy( Col, Row);
   WRITE( Message);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Draw_Box( Row1, Row2, Col1, Col2 : BYTE);

CONST Topleft = #201;
      Botleft = #200;
           Up = #186;
       Across = #205;
     Topright = #187;
     Botright = #188;

VAR Pos : INTEGER;

BEGIN
  Gotoxy( Col1, Row1);
  WRITE( Topleft);
  FOR Pos:= (Col1 + 1) TO (Col2 - 1) DO
      WRITE( Across);
  WRITE( Topright);
  FOR Pos:= (Row1 + 1) TO (Row2 - 1) DO BEGIN
      Gotoxy( Col1, Pos);
      WRITE( Up);
      Gotoxy( Col2, Pos);
      WRITE( Up);
   END;
   Gotoxy( Col1, Row2);
   WRITE( Botleft);
   FOR Pos:= (Col1 + 1) TO (Col2 - 1) DO
       WRITE( Across);
   WRITE( Botright);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Init_Graph;

BEGIN
   Graphdriver:= Detect;
   Initgraph( Graphdriver, Graphmode, 'C:DRIVERS');
   Errorcode:= Graphresult;
   IF (Errorcode <> Grok) THEN BEGIN
      WRITELN('Graphics problem - ',Grapherrormsg( Errorcode));
      WRITELN('Program aborted');
      Halt(1);
   END
   ELSE BEGIN
      Setgraphmode( Graphmode);
      Settextstyle( Smallfont, Horizdir, 6);
  END;
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE End_Graph;

BEGIN
  Restorecrtmode;
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------------}
{Graphic hardcopy routine - modified from that for Turbo 3.0 presented}
{in the TurboPascal Grafbox.                                             }

VAR Colorglb : BYTE;
    Xmaxglb, Xscreenmaxglb, Ymaxglb : INTEGER;


{-------------------------------------------------------------------------}
FUNCTION Checkcol( X, Y : INTEGER) : BOOLEAN;  {Colour at x,y = ColourGlb?}

BEGIN
    Checkcol:= (Getpixel( X,Y) <> Black)
    {NB! - this replacement of the original}
    {code in Turbo 3.0 GrafBox is VITAL for}
    {the routine to work with version 4.0. }
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE hardcopy(inverse:BOOLEAN;mode:BYTE); { EPSON only!}
                                               {  - based on Turbo 3 Grafbox}
  VAR i,j,top:INTEGER;
      Colorloc,Printbyte:BYTE;

  PROCEDURE doline(top:INTEGER);
     VAR M : INTEGER;

    FUNCTION Constructbyte(j,i:INTEGER):BYTE;
      CONST Bits:ARRAY [0..7] OF BYTE=(128,64,32,16,8,4,2,1);
      VAR Cbyte,k:BYTE;
      BEGIN
        i:=i shl 3;
        Cbyte:=0;
        FOR k:=0 TO top DO
          IF Checkcol(j,i+k) THEN Cbyte:=Cbyte OR Bits[k];
        Constructbyte:=Cbyte;
      END;

  BEGIN              {Of DOLINE}
      IF mode=1 THEN WRITE(lst,^['L')
      ELSE WRITE(lst,^['*',CHR(mode));
      WRITE(lst,CHR(lo(Xscreenmaxglb+1)),CHR(Hi(Xscreenmaxglb+1)));
      FOR M:=0 TO Xscreenmaxglb DO
       BEGIN
        Printbyte:=Constructbyte(M,i);
        IF inverse THEN Printbyte:=NOT Printbyte;
        WRITE(lst,CHR(Printbyte));
       END;
      IF mode<>4 THEN WRITELN(lst);
    END;


BEGIN
    Xscreenmaxglb:= Getmaxx;
    Xmaxglb:= (Getmaxx + 1) DIV 8 - 1;
    Ymaxglb:= Getmaxy;
    top:=7;
    Colorloc:=Colorglb;
    Colorglb:=255;
    mode:=mode AND 7;
    IF (mode=5) OR (mode=0) THEN mode:=4;
    WRITE(lst,^['3'#24);
    FOR i:=0 TO ((Ymaxglb+1) shr 3)-1 DO doline(7);
    i:=((Ymaxglb+1) shr 3);
    IF (Ymaxglb+1) AND 7<>0 THEN
      doline((Ymaxglb+1) AND 7);
    WRITELN(lst,^['2');
    Colorglb:=Colorloc;
END;
{-------------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Cursoff;

BEGIN
   Regis.Ax:= $0100;
   Regis.Cx:= $2000;
   Intr( $10, Regis);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Curson;

BEGIN
  Regis.Ax:= $0100;
  Regis.Cx:= $0607;
  Intr( $10, Regis)
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Pause;
  VAR Any_Old_Arbitrary_Key : CHAR;

BEGIN
   Any_Old_Arbitrary_Key:= Readkey;
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Alarm;
{Sends 'bells' on fatal error to alert user.}
  VAR I, J : INTEGER;

BEGIN
   FOR I:= 1 TO 5 DO BEGIN
       WRITE(CHR(7));
       FOR J:= 1 TO 4000 DO {Delay};
   END
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Clear_From( Screen_Line : BYTE);
{Clears the screen from SCREEN_LINE down, and leaves cursor}
{at the beginning of SCREEN_LINE.                          }
   VAR I : BYTE;

BEGIN
   FOR I:= Screen_Line TO 25 DO BEGIN
       Gotoxy(1, I);
       Clreol
   END;
   Gotoxy(1, Screen_Line)
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Drawlogo;
{Needs file 'LOG.NUD' as well as .CHR character files.}

TYPE Charset = SET OF CHAR;

VAR Curinfil, Numr, Numc : INTEGER;
     Ch_Scale, Pic_Scale : ARRAY[ 0..255] OF INTEGER;
         Curx, Cury, Num : INTEGER;
                Size, W : WORD;
                   Fset : Charset;
                    Ch : CHAR;
                     P : Pointer;
                     I : WORD;


{-------------------------------------------------------------------}
PROCEDURE Get_Picture_Scale;

BEGIN
   Pic_Scale[ORD(' ')]:= White;
   Pic_Scale[ORD('''')]:= Lightgray;
   Pic_Scale[ORD('.')]:= Lightred;
   Pic_Scale[ORD(':')]:= Yellow;
   Pic_Scale[ORD(';')]:= Yellow;
   Pic_Scale[ORD(']')]:= Lightgray;
   Pic_Scale[ORD('[')]:= Lightgray;
   Pic_Scale[ORD('/')]:= Blue;
   Pic_Scale[ORD('-')]:= Darkgray;
   Pic_Scale[ORD('A')]:= Cyan;
   Pic_Scale[ORD('B')]:= Red;
   Pic_Scale[ORD('D')]:= Magenta;
   Pic_Scale[ORD('F')]:= Cyan;
   Pic_Scale[ORD('H')]:= Brown;
   Pic_Scale[ORD('I')]:= Brown;
   Pic_Scale[ORD('M')]:= Lightgray;
   Pic_Scale[ORD('P')]:= Blue;
   Pic_Scale[ORD('V')]:= Yellow;
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Get_And_Draw_File;
  VAR Colr : WORD;
      Infil : TEXT;
      I : INTEGER;

BEGIN
  Get_Picture_Scale;
  ASSIGN( Infil, 'LOG.NUD');
  RESET( Infil);
  Curx:= 2; Cury:= 379;
  WHILE (NOT(EOF(Infil))) DO BEGIN
      WHILE( NOT(EOLN(Infil))) DO BEGIN
            READ( Infil, Ch);
            Colr:= Pic_Scale[ ORD(Ch)];
            Putpixel( Curx, Cury, Colr);
            Curx:= Curx + 1;
      END;
      Cury:= Cury + 1;
      Curx:= 2;
      READLN( Infil);
  END;
  CLOSE( Infil);
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Draw_Icons;

BEGIN
  Get_And_Draw_File;
  Setcolor( White);
  Size:= Imagesize( 2, 379, 87, 479);
  Getmem( P, Size);
  Getimage( 2, 379, 87, 479, P^);
  Putimage( 2, 2, P^, Normalput);
  Putimage( 553, 2, P^, Normalput);
  Putimage( 553, 380, P^, Normalput);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Draw_Inner;
   VAR Col, Repeats : WORD;

BEGIN
  Setfillstyle( Solidfill, Lightred);
  Setcolor( White);
  Bar3d( 200, 150, 442, 330, 0, Topon);
  Setcolor( Darkgray);
  Settextstyle(defaultfont, Horizdir, 2);
  Outtextxy( 218, 155, 'FIRM SOFTWARE');
  Settextstyle(defaultfont, Horizdir, 2);
  Outtextxy( 300, 265,'By');
  Outtextxy( 204, 295, 'Stuart Melville');
  Arc( 321, 230, 0, 360, 20);
  Col:= 1;
  WHILE (NOT Keypressed) DO BEGIN
        Setfillstyle( Solidfill, Col);
        Floodfill( 321, 230, Darkgray);
        Delay( Random( 100));
        Col:= Col + 1;
        IF (Col = Darkgray) THEN
           Col:= Col + 1;
        IF (Col = 16) THEN
           Col:= 1;
  END;
  Ch:= Readkey;
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Drawlogo;
   VAR I, J : INTEGER;

BEGIN
  Draw_Icons;
  Draw_Inner;
END;
{-------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------}
PROCEDURE Set_Background;

BEGIN
  Setbkcolor( Lightgray);
  Setcolor( White);
  Cleardevice;
  Line( 1, 1, 1, 480);
  Line( 639, 1, 639, 480);
  Line( 1, 1, 640, 1);
  Line( 1, 480, 640, 480);
  Line( 1, 479, 640, 479);
  Line( 320, 1, 320, 480);
  Line( 1, 240, 640, 240);
  Setfillstyle( Solidfill, Red);
  Floodfill( 5,5, White);
  Setfillstyle( Solidfill, Cyan);
  Floodfill( 420, 5, White);
  Setfillstyle( Solidfill, Blue);
  Floodfill( 5, 420, White);
  Setfillstyle( Solidfill, Green);
  Floodfill( 420, 420, White);
END;
{-------------------------------------------------------------------}


BEGIN
  Init_Graph;
  Set_Background;
  FOR Curx:= 2 TO 87 DO
     FOR Cury:= 379 TO 479 DO
         Putpixel( Curx, Cury, White);
  Curx:= 1; Cury:= 1;
  Drawlogo;
  End_Graph;
END;
{-------------------------------------------------------------------}

END.
```

Appendix J - Source Listing of the Datcom_Abel Program

(Based on Formula 1)

```
{$N+}     {8087 on board}
{$R-}     {Range checking off}
{$B+}     {Boolean complete evaluation on}
{$S+}     {Stack checking on}
{$I+}     {I/O checking on}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}

PROGRAM Datacom_By_Abel;
{Checks datacom by Abel's formula of R = (Pr/(N0*(Eb/N0)).}


USES
  Crt,
  Dos;

CONST  File_Table_Size       = 100;
       Modelled_Trail_Types : SET OF BYTE = [];

       Printer = TRUE;
       Console = FALSE;

       Number_Of_Iterations = 3;     { Search depth    }
       Bit_Index_Range      = 4;     {     "      "    }
       Block_Index_Range    = 4;     {     "      "    }

       Speed_Of_Light       = 300000.0;

       Max_Iter             = 5;     {Max iterations.}
       debug                = FALSE;
       Ln_10                = 2.3025851;
       Exp_Cutoff           = -88.0;

TYPE   Unsigned = ARRAY [1..2] OF BYTE;

       Str80          = String [80];
       Str40          = String [40];
       Str20          = String [20];
       Str14          = String [14];
       Str12          = String [12];
       Str10          = String [10];
       Str5           = String [5];
       Str2           = String [2];

       Time_Rec = RECORD
          Hours, Mins, Secs, Huns  : BYTE
       END;

       Dbs_Over_Noise = (Plus6, Plus9, Plus12, Plus15);

       Trail_Header = RECORD
          Trail_Num              : INTEGER;
          Start_Time             : Time_Rec;
          Elapsed_Time           : REAL;
          Num_Samples            : INTEGER;
          Sample_Range           : ARRAY [0..1] OF REAL;
          Noise_Level            : INTEGER;
          Number_Of_Fades        : INTEGER;
          Peak_Time              : INTEGER;
          Peak_Strength          : INTEGER;
          Signal_Present_Time    : ARRAY [0..1] OF Unsigned;
          Useful_Time            : ARRAY [Plus6..Plus15] OF Unsigned;
          Trail_Type             : BYTE;
          Sample_Mean            : REAL;
          Matched_With           : INTEGER;
          Spare_1, Spare_2       : INTEGER;
          Errors_Found           : BYTE
       END;
```

```
      Header_Fil    = FILE OF Trail_Header;
      Sample_Fil    = FILE OF BYTE;

      Data_Array    = ARRAY [0..10000] OF INTEGER;
      Numset        = SET OF 2..29;
      Charset       = SET OF CHAR;


VAR   File_Table                          : ARRAY [1..File_Table_Size] OF Str12;
      Number_Files_In_Table               : INTEGER;
      Header_File                         : Header_Fil;
      Sample_File                         : Sample_Fil;
      Hrec                                : Trail_Header;
      Trail_Array                         : Data_Array;
      Probe_Length, Ack_Length            : INTEGER;
      Bandwidth, Crc_Length               : INTEGER;
      Data_Bits_Per_Block                 : REAL;
      Olddir, Path : String;
      D : INTEGER;
      Probe_Time, Ack_Size_Xsecs          : REAL;
      Propagation_Delay, Path_Length      : REAL;
      Block_Size_Xsecs                    : REAL;
      Bits_Per_Xsec, Sync_Xsecs           : REAL;
      Anykey                              : CHAR;
      Bit_Value                           : ARRAY [1..Max_Iter,1..Bit_Index_Range]
                                                                    OF REAL;
      Block_Value                         : ARRAY [1..Max_Iter,1..Block_Index_Range]
                                                                    OF REAL;
      Comms_Info                          : ARRAY [1..Max_Iter,1..Bit_Index_Range,
                                                   1..Block_Index_Range,
                                                   1..2] OF REAL;
      Sig_Noise_On, Sig_Noise_Off         : REAL;
      On_Bit_Err_Rate                     : REAL;
      Off_Bit_Err_Rate                    : REAL;
      Switch_From_Transmit_Xsecs          : REAL;
      Switch_From_Receive_Xsecs           : REAL;
      Outf                                : TEXT;
      Sink                                : BOOLEAN;
      Trail_Dbm_Alteration                : INTEGER;
      Transmit_Power, Measured_Power      : INTEGER;
      User_Block_Size, User_Bit_Rate      : REAL;
      Optimising                          : BOOLEAN;
      Tystr                               : Str20;
      Types_Wanted                        : Numset;
      Match_Only, No_Matches              : BOOLEAN;
      Num_Trails_Considered               : REAL;
      Match_Str                           : Str40;
      Remote_Time_Wanted                  : BOOLEAN;
      Trans_Xsecs                         : REAL;
      N0, Eb_To_N0, Pr, Bitr              : REAL;
      Bitsum                              : REAL;
      File_Used : INTEGER;
      Akey : CHAR;
      Rates_Array  : ARRAY[1..1000] OF WORD;


{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
PROCEDURE debugwrite( st : string);
 VAR akey : CHAR;
BEGIN
   WRITELN( st);
   akey:= readkey;
END;

{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Menu_Head;

BEGIN
  Clrscr;
  WRITE ('METEORS - Data-comm Simulation          ');
  WRITE ('Written by  SW Melville')
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Menu_Write( St_In : Str80);
{Writes a string in 'menu' fashion. Does this by highlighting first}
{letter of string, (the option character). eg. If passed the string}
{"A =  Abort program", it would print out a  highlighted "A",  then}
{the rest of the string at lower intensity.                          }

BEGIN
   Normvideo;
   WRITE(St_In[1]);
   St_In[1]:= ' ';
   Lowvideo;
   WRITELN(St_In);
   Normvideo;
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
FUNCTION Get_Valid_Choice_From( Ok_Chars : Charset) : CHAR;
   VAR Choice : CHAR;

BEGIN
   REPEAT
      Choice:= Upcase( Readkey)
   UNTIL (Choice IN Ok_Chars);
   Get_Valid_Choice_From:= Choice
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
FUNCTION Users_Response_To (Chars  : INTEGER; Question  : Str80) :Str12;
VAR  Answer       : Str12;
     Pos          : BYTE;
     Ans          : CHAR;

BEGIN
  Gotoxy (2,22);
  WRITE (Question, '  ');
  IF Chars = 1 THEN          { Read only one character }
   BEGIN
     Ans:= Readkey;
     Answer := Ans
   END
  ELSE                       { Read until user presses RETURN }
     READLN (Answer);

  FOR Pos := 1 TO LENGTH (Answer) DO
    IF (ORD (Answer[Pos]) >= 97) AND (ORD (Answer[Pos]) <= 122) THEN
      Answer[Pos] := CHR (ORD (Answer[Pos]) - 32);

  Users_Response_To := Answer;
  Gotoxy (2,22);   { Clear the question and answer from the screen }
  Clreol
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
FUNCTION Get_User_Int (Question          : Str80;
                       Min_Int, Max_Int  : INTEGER)  : INTEGER;

VAR  Answer, Err      : INTEGER;
     Answer_Str       : Str12;

BEGIN
  REPEAT
    Answer_Str := Users_Response_To (6, Question);
    VAL (Answer_Str, Answer, Err);
  UNTIL (Err = 0) AND (Answer >= Min_Int) AND (Answer <= Max_Int);
  Get_User_Int := Answer
END;
{-------------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Float (INT  : INTEGER)  : REAL;

BEGIN
  Float := INT
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Fill_File_Table;

{ This procedure will fill the file table with all the entries  }
{ in the current directory whose system is specified by System_ }
{ Char.                                                         }
   VAR Dirinfo : Searchrec;
          Junk : String;
             D : INTEGER;
BEGIN
  Number_Files_In_Table := 0;
  Findfirst( 'H*.*', Anyfile, Dirinfo);
  WHILE (Doserror = 0) DO
   BEGIN
     IF (Dirinfo.Name[2] IN ['D','N']) THEN BEGIN
           Number_Files_In_Table := Number_Files_In_Table + 1;
           File_Table [Number_Files_In_Table] := Dirinfo.Name;
ber_files_in_tabl WRITELN(     number_files_in_table,'     ',file_table[num-
           akey:= readkey;
     END;
     Findnext( Dirinfo);
   END;
   IF Number_Files_In_Table >= File_Table_Size THEN BEGIN
        Number_Files_In_Table:= File_Table_Size;
        Junk := Users_Response_To (1, 'File table overflow - ' +
          'only the first 100 files will be considered (OK)?')
   END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE List_File_Table_To_Screen;

{ List all header files in the table to the screen. }

VAR  File_Number         : INTEGER;

BEGIN
  Clrscr;
  Menu_Head;
  Gotoxy (1, 5);
  WRITELN ('Meteor header files on current directory are :');
  WRITELN;
  FOR File_Number := 1 TO Number_Files_In_Table DO
     WRITE ('  [', File_Number :2, '] ', File_Table [File_Number], '    ')
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}

{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Close_Files;

BEGIN
  CLOSE (Header_File);
  CLOSE (Sample_File)
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Open_Header_File (     File_Name     : Str14;
                            VAR  File_Var      : Header_Fil;
                            VAR  File_Present  : BOOLEAN);

VAR  Junk       : String;

BEGIN
  ASSIGN (File_Var, File_Name);
  {$I-}
  RESET (File_Var);
  {$I+}
  File_Present := Ioresult = 0;
  IF NOT File_Present THEN
     Junk := Users_Response_To (1, 'File ' + File_Name +
               ' not on drive (OK)?');
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Open_Sample_File (     File_Name     : Str14;
                            VAR  File_Var      : Sample_Fil;
                            VAR  File_Present  : BOOLEAN);

VAR  Junk        : String;

BEGIN
  ASSIGN (File_Var, File_Name);
  {$I-}
  RESET (File_Var);
  {$I+}
  File_Present := Ioresult = 0;
  IF NOT File_Present THEN
     Junk := Users_Response_To (1, 'File ' + File_Name +
               ' not on drive (OK)?');
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Ready_Files (     Pos_In_Table     : INTEGER;
                       VAR  Files_Availible  : BOOLEAN);

VAR  File_Present       : BOOLEAN;
     Sample_Name        : Str12;


BEGIN
  Open_Header_File (File_Table [Pos_In_Table], Header_File, File_Present);
  IF File_Present THEN
   BEGIN
     Sample_Name := File_Table [Pos_In_Table];
     Sample_Name [1] := 'S';
     Open_Sample_File (Sample_Name, Sample_File, File_Present);
     IF File_Present THEN  Files_Availible := TRUE
     ELSE                  Files_Availible := FALSE
   END
  ELSE  Files_Availible := FALSE;
END;
{-----------------------------------------------------------------------}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Get_Next_Trail;

CONST Bad_Data = 1;

VAR   Sample_Gap, Sample          : INTEGER;
      Real_Gap                    : REAL;
      Sample_Value                : BYTE;
      Longval : LONGINT;

BEGIN
  READ (Header_File, Hrec);
  Real_Gap := (Hrec.Sample_Range [1] - Hrec.Sample_Range [0]) + 1;
  IF ((Real_Gap > 10000.0) OR (Real_Gap < 1.0))
      OR (Hrec.Noise_Level > 60) OR (Hrec.Noise_Level < 0)
      OR (Hrec.Trail_Type = Bad_Data) THEN BEGIN
      {STUFFED_DATA!! - set trailarray[0] to 0 to avoid crash.}
      Trail_Array[0]:= 0;
      Gotoxy(2,22);
      WRITE('Trail ',Hrec.Trail_Num,' was ill.');
      WRITE(' - ',Hrec.Num_Samples,' Samples,  Noise = ',Hrec.Noise_Level-140)
  END
  ELSE BEGIN
      Num_Trails_Considered:= Num_Trails_Considered + 1.0;
      Sample_Gap:= TRUNC (Real_Gap);
      Longval:= TRUNC(Hrec.Sample_Range[0]);
      SEEK (Sample_File, Longval);
      FOR Sample := 1 TO Sample_Gap DO
      BEGIN
         READ (Sample_File, Sample_Value);
         Trail_Array [Sample] := Sample_Value - 140
      END;
      Trail_Array [0] := Sample_Gap
 END
END;
{-----------------------------------------------------------------------}



{-----------------------------------------------------------------------}
FUNCTION Real_Round (Reel   : REAL)   : REAL;

BEGIN
  Reel := Reel + 0.5;
  Real_Round := Reel - Frac (Reel)
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
FUNCTION Dbm_Scaled( Val_In : INTEGER) : REAL;
   VAR Prec : Double;

BEGIN
   Prec:= Exp( (Val_In/10.0) * Ln_10);
   Dbm_Scaled:= Prec;
END;
{-----------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Compute_Info_For_Files (Start_File, Stop_File : INTEGER);
  VAR Files_Availible : BOOLEAN;
          File_Number : INTEGER;


PROCEDURE Update_Bit_Info;
  VAR N0_Times_Ratio, Bits_Sent : REAL;
                      Trailpos : INTEGER;
                         Kbits : WORD;

BEGIN
  N0:= Dbm_Scaled( ( Hrec.Noise_Level - 140));
  N0:= N0/2000.0;                          {Bandwidth normalisation.}
  N0_Times_Ratio:= N0 * Eb_To_N0;
  FOR Trailpos:= 1 TO Trail_Array[0] DO BEGIN
      Pr:= Dbm_Scaled( Trail_Array[ Trailpos]);
      Bitr:= Pr/N0_Times_Ratio;
      {Now have best bitrate for this sample, see how many can send in 5 mS.}
      Bits_Sent:= Bitr/200.0;     {1000/5 = 200 => 1000/200 = 5!}
      Bitsum:= Bitsum + Bits_Sent;
      Kbits:= ROUND( Bitr/1000.0);
      IF (Kbits > 1000) THEN
          Kbits:= 1000;
      IF (Kbits < 1) THEN
          Kbits:= 1;
      Rates_Array[ Kbits]:= Rates_Array[ Kbits] + 1;
  END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


BEGIN
  Menu_Head;
  FOR File_Number := Start_File TO Stop_File DO BEGIN
      Ready_Files (File_Number, Files_Availible);
      IF Files_Availible THEN BEGIN
         Gotoxy (5, 7);
         WRITE ('Computing bit rate for file ', File_Table [File_Number]);
         Clreol;
         Gotoxy (5, 9);
         WRITE ('Processing trail ');
         Clreol;
         WHILE NOT (EOF (Header_File)) DO BEGIN
            REPEAT
               Get_Next_Trail;
            UNTIL ((Trail_Array[0] <> 0) OR (EOF( Header_File)));
            IF (Trail_Array[0] <> 0) THEN BEGIN
               Gotoxy (22, 9);
               WRITE (Hrec.Trail_Num);
               Update_Bit_Info;
            END
         END;
      END
      ELSE WRITELN('No files found.');
  END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{---------------------------------------------------------------------}
PROCEDURE Auto_Process_Files_In_Directory;

VAR  Junk                                                  : String;
     Start_File, Stop_File                                 : INTEGER;
     Number_Bits_Across, Best_Block_Size, Best_Bit_Rate    : REAL;
     Bit_Index, Block_Index                                : INTEGER;

BEGIN
  Fill_File_Table;
  IF Number_Files_In_Table = 0 THEN
     Junk := Users_Response_To (1, 'No meteor data files found (OK)?')
  ELSE
   BEGIN
     List_File_Table_To_Screen;
     Start_File := Get_User_Int ('Enter start file number :',
        1, Number_Files_In_Table);
     file_used:= start_file;
     WRITELN( file_used);
     akey:= readkey;
     Stop_File := Get_User_Int ('Enter stop file number :',
        1, Number_Files_In_Table);
     Num_Trails_Considered:= 0.0;
     Compute_Info_For_Files (Start_File, Stop_File);
   END
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
PROCEDURE Get_Path;
   VAR Newpath : String;

BEGIN
  Path:= 'C:\WORK\METEOR\DATA';
  Getdir( 0, Olddir);
  REPEAT
    WRITE('Please enter path (default is \work\meteor\data) : ');
    READLN( Newpath);
    IF (Newpath = '') THEN
       Newpath:= Path;
    {$I-} Chdir( Newpath); {$I+}
  UNTIL (Ioresult = 0);
  Path:= Newpath;
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
PROCEDURE Get_Eb;

BEGIN
   Clrscr;
   WRITE('Please enter Eb to N0 ratio needed (Energy per bit to noise) : ');
   READLN( Eb_To_N0)
END;
{---------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Show_Results;
  VAR Int_Bitsum, I, Rat : LONGINT;

BEGIN
  ASSIGN( Outf, 'BITRES.PRT');
  REWRITE( Outf);
  WRITELN( Outf, 'File used : ',File_Table[ File_Used]);
  WRITELN( Outf, 'Ratio chosen was ',Eb_To_N0:3:1,' : 1');
  WRITE( Outf, 'Total bits transferred in the hour was : ');
  Int_Bitsum:= ROUND( Bitsum);
  WRITELN( Outf, Int_Bitsum);
  WRITELN( Outf);
  WRITELN( Outf, '     BIT RATE INFORMATION');
  WRITELN( Outf, '     ~~~~~~~~~~~~~~~~~~~');
  WRITELN( Outf, 'Rate (Kbits/sec)   Time used          Bits sent          %');
  FOR I:= 1 TO 1000 DO BEGIN
     Rat:= Rates_Array[ I];
     IF (Rat <> 0) THEN BEGIN
         WRITE( Outf, I:10, ' ':9);
         WRITE( Outf, ((Rat * 5)/1000):6:3,' secs         ');
         WRITE( Outf, (Rat * I * 5.0):8:0);
         {In Kbits and mSecs so 1000s balance, * 5 for sample spacing.}
         WRITELN( Outf, ' ',(((Rat * I * 5)/Bitsum) * 100.0):10:1);
     END;
  END;
  CLOSE( Outf);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


BEGIN                    {Main Program}
  Get_Path;
  Get_Eb;
  Fillchar( Rates_Array, Sizeof( Rates_Array), 0);
  Bitsum:= 0.0;
  Auto_Process_Files_In_Directory;
  Show_Results;
END.
```

Appendix K - Source Listing of Program Datacom

(Half-Duplex Simulation)

```
{$N+}     {8087 on board}
{$R-}     {Range checking off}
{$B+}     {Boolean complete evaluation on}
{$S+}     {Stack checking on}
{$I+}     {I/O checking on}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}

PROGRAM Data_Comm_Simulation;

{ This program will either optimise the block size and bit rate }
{ for a given link, or simply compute the throughput for a user }
{ specified block size and bit rate.                            }
{        By Stuart Melville & Robert Letschert, 1987.           }
{           Last modified : December 1990                       }


USES
  Crt,
  Dos;

CONST  File_Table_Size      = 100;
       Modelled_Trail_Types : SET OF BYTE = [];

       Printer = TRUE;
       Console = FALSE;

       Number_Of_Iterations = 3;     { Search depth   }
       Bit_Index_Range      = 4;     {     "      "    }
       Block_Index_Range    = 4;     {     "      "    }

       Speed_Of_Light       = 300000.0;

       Max_Iter             = 5;     {Max iterations.}
       debug                = FALSE;
       Ln_10                = 2.3025851;
       Exp_Cutoff           = -88.0;

      Bellnum = 1;
      U_Densenum = 2;
      O_Densenum = 3;
      Mushnum = 4;
      Gothicnum = 5;

TYPE  Unsigned = ARRAY [1..2] OF BYTE;

      Str80          = String [80];
      Str40          = String [40];
      Str20          = String [20];
      Str14          = String [14];
      Str12          = String [12];
      Str10          = String [10];
      Str5           = String [5];
      Str2           = String [2];

      Time_Rec = RECORD
         Hours, Mins, Secs, Huns  : BYTE
      END;

      Dbs_Over_Noise = (Plus6, Plus9, Plus12, Plus15);
```

```
      Trail_Header = RECORD
         Trail_Num              : INTEGER;
         Start_Time             : Time_Rec;
         Elapsed_Time           : REAL;
         Num_Samples            : INTEGER;
         Sample_Range           : ARRAY [0..1] OF REAL;
         Noise_Level            : INTEGER;
         Number_Of_Fades        : INTEGER;
         Peak_Time              : INTEGER;
         Peak_Strength          : INTEGER;
         Signal_Present_Time    : ARRAY [0..1] OF Unsigned;
         Useful_Time            : ARRAY [Plus6..Plus15] OF Unsigned;
         Trail_Type             : BYTE;
         Sample_Mean            : REAL;
         Matched_With           : INTEGER;
         Spare_1, Spare_2       : INTEGER;
         Errors_Found           : BYTE
      END;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
      Header_Fil    = FILE OF Trail_Header;
      Sample_Fil    = FILE OF BYTE;

      Data_Array    = ARRAY [0..10000] OF INTEGER;

      Numset        = SET OF 2..29;
      Charset       = SET OF CHAR;
      Trailset      = SET OF 1..30;

      Typearr = ARRAY[0..30] OF REAL;

VAR   File_Table                        : ARRAY [1..File_Table_Size] OF Str12;
      Number_Files_In_Table             : INTEGER;
      Header_File                       : Header_Fil;
      Sample_File                       : Sample_Fil;
      Hrec                              : Trail_Header;
      Trail_Array                       : Data_Array;
      Probe_Length, Ack_Length          : INTEGER;
      Bandwidth, Crc_Length             : INTEGER;
      Data_Bits_Per_Block               : REAL;
      Olddir, Path : String;
      D : INTEGER;
      Probe_Time, Ack_Size_Xsecs        : REAL;
      Propagation_Delay, Path_Length    : REAL;
      Block_Size_Xsecs                  : REAL;
      Bits_Per_Xsec, Sync_Xsecs         : REAL;
      Anykey                            : CHAR;
      Bit_Value                         : ARRAY [1..Max_Iter,1..Bit_Index_Range]
                                                                  OF REAL;
      Block_Value                       : ARRAY [1..Max_Iter,1..Block_Index_Range]
                                                                  OF REAL;
      Comms_Info                        : ARRAY [1..Max_Iter,1..Bit_Index_Range,
                                            1..Block_Index_Range,
                                            1..2] OF REAL;
      Sig_Noise_On, Sig_Noise_Off       : REAL;
      On_Bit_Err_Rate                   : REAL;
      Off_Bit_Err_Rate                  : REAL;
      Switch_From_Transmit_Xsecs        : REAL;
      Switch_From_Receive_Xsecs         : REAL;
      Outf                              : TEXT;
      Sink                              : BOOLEAN;
      Trail_Dbm_Alteration              : INTEGER;
      Transmit_Power, Measured_Power    : INTEGER;
      User_Block_Size, User_Bit_Rate    : REAL;
      Optimising                        : BOOLEAN;
      Tystr                             : Str20;
      Types_Wanted                      : Numset;
      Match_Only, No_Matches            : BOOLEAN;
      Num_Trails_Considered             : REAL;
      Match_Str                         : Str40;
      Remote_Time_Wanted                : BOOLEAN;
      Trans_Xsecs                       : REAL;
      Start_Block, Stop_Block           : REAL;
      Bells, U_Dense, O_Dense, Mush, Gothic : Trailset;
```

```
      Start_Bit, Stop_Bit              : REAL;
      Typecount                        : Typearr;

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE debugwrite( st : string);
 VAR akey : CHAR;
BEGIN
   WRITELN( st);
   akey:= readkey;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Menu_Head;

BEGIN
  Clrscr;
  WRITE ('METEORS - Data-comm Simulation          ');
  WRITE ('Written by  S. Melville and R. Letschert')
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Menu_Write( St_In : Str80);
{Writes a string in 'menu' fashion. Does this by highlighting first}
{letter of string, (the option character). eg. If passed the string}
{"A =  Abort program", it would print out a  highlighted "A",  then}
{the rest of the string at lower intensity.                        }

BEGIN
   Normvideo;
   WRITE(St_In[1]);
   St_In[1]:= ' ';
   Lowvideo;
   WRITELN(St_In);
   Normvideo;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Get_Valid_Choice_From( Ok_Chars : Charset) : CHAR;
   VAR Choice : CHAR;

BEGIN
   REPEAT
      Choice:= Upcase( Readkey)
   UNTIL (Choice IN Ok_Chars);
   Get_Valid_Choice_From:= Choice
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Users_Response_To (Chars   : INTEGER; Question   : Str80) :Str12;

{-----------------------------------------------------------------------}
{ This function takes as input a question stored in QUESTION, and      }
{ produces as output the user's response to the question. If CHARS     }
{ is 1 then one char is read and no RETURN is necessary from the       }
{ user. If anything else, the user must press RETURN after his/her     }
{ answer. The reason this was included as an INTEGER variable and      }
{ not a BOOLEAN is primarily for extension purposes. The answer is     }
{ automatically mapped to upper-case in both instances.                }
{-----------------------------------------------------------------------}

VAR  Answer       : Str12;
     Pos          : BYTE;
     Ans          : CHAR;

BEGIN
  Gotoxy (2,22);
  WRITE (Question, '  ');
  IF Chars = 1 THEN           { Read only one character }
   BEGIN
     Ans:= Readkey;
     Answer := Ans
   END
  ELSE                        { Read until user presses RETURN }
     READLN (Answer);
  FOR Pos := 1 TO LENGTH (Answer) DO
    IF (ORD (Answer[Pos]) >= 97) AND (ORD (Answer[Pos]) <= 122) THEN
      Answer[Pos] := CHR (ORD (Answer[Pos]) - 32);

  Users_Response_To := Answer;
  Gotoxy (2,22);    { Clear the question and answer from the screen }
  Clreol
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Get_User_Int (Question              : Str80;
                       Min_Int, Max_Int      : INTEGER)   : INTEGER;

VAR  Answer, Err        : INTEGER;
     Answer_Str         : Str12;

BEGIN
  REPEAT
    Answer_Str := Users_Response_To (6, Question);
    VAL (Answer_Str, Answer, Err);
  UNTIL (Err = 0) AND (Answer >= Min_Int) AND (Answer <= Max_Int);
  Get_User_Int := Answer
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Float (INT   : INTEGER)   : REAL;

BEGIN
  Float := INT
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Fill_File_Table;

{ This procedure will fill the file table with all the entries  }
{ in the current directory whose system is specified by System_ }
{ Char.                                                         }
   VAR Dirinfo : Searchrec;
         Junk : String;
            D : INTEGER;
{-----------------------------------------------------------------------}
BEGIN
  Number_Files_In_Table := 0;
  Findfirst( 'H*.*', Anyfile, Dirinfo);
  WHILE (Doserror = 0) DO
   BEGIN
    IF (Dirinfo.Name[2] IN ['D','N']) THEN BEGIN
          Number_Files_In_Table := Number_Files_In_Table + 1;
          File_Table [Number_Files_In_Table] := Dirinfo.Name;
    END;
    Findnext( Dirinfo);
   END;
   IF Number_Files_In_Table >= File_Table_Size THEN BEGIN
        Number_Files_In_Table:= File_Table_Size;
        Junk := Users_Response_To (1, 'File table overflow - ' +
          'only the first 100 files will be considered (OK)?')
   END;
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE List_File_Table_To_Screen;

{ List all header files in the table to the screen. }

VAR  File_Number          : INTEGER;

BEGIN
  Clrscr;
  Menu_Head;
  Gotoxy (1, 5);
  WRITELN ('Meteor header files on current directory are :');
  WRITELN;
  FOR File_Number := 1 TO Number_Files_In_Table DO
     WRITE ('   [', File_Number :2, '] ', File_Table [File_Number], '    ')
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Close_Files;

BEGIN
  CLOSE (Header_File);
  CLOSE (Sample_File)
END;
{-----------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Open_Header_File (     File_Name      : Str14;
                            VAR  File_Var       : Header_Fil;
                            VAR  File_Present   : BOOLEAN);

VAR  Junk       : String;

BEGIN
  ASSIGN (File_Var, File_Name);
  {$I-}
  RESET (File_Var);
  {$I+}
  File_Present := Ioresult = 0;
  IF NOT File_Present THEN
     Junk := Users_Response_To (1, 'File ' + File_Name +
             ' not on drive (OK)?');
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Open_Sample_File (     File_Name      : Str14;
                            VAR  File_Var       : Sample_Fil;
                            VAR  File_Present   : BOOLEAN);

VAR  Junk        : String;

BEGIN
  ASSIGN (File_Var, File_Name);
  {$I-}
  RESET (File_Var);
  {$I+}
  File_Present := Ioresult = 0;
  IF NOT File_Present THEN
     Junk := Users_Response_To (1, 'File ' + File_Name +
             ' not on drive (OK)?');
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Ready_Files (     Pos_In_Table     : INTEGER;
                       VAR  Files_Availible  : BOOLEAN);

VAR  File_Present          : BOOLEAN;
     Sample_Name           : Str12;


BEGIN
  Open_Header_File (File_Table [Pos_In_Table], Header_File, File_Present);
  IF File_Present THEN
    BEGIN
      Sample_Name := File_Table [Pos_In_Table];
      Sample_Name [1] := 'S';
      Open_Sample_File (Sample_Name, Sample_File, File_Present);
      IF File_Present THEN  Files_Availible := TRUE
      ELSE                  Files_Availible := FALSE
    END
  ELSE  Files_Availible := FALSE;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Get_Next_Trail;

CONST Bad_Data = 1;

VAR  Sample_Gap, Sample          : INTEGER;
     Real_Gap                    : REAL;
     Sample_Value                : BYTE;
     Longval : LONGINT;

BEGIN
  READ (Header_File, Hrec);
  Real_Gap := (Hrec.Sample_Range [1] - Hrec.Sample_Range [0]) + 1;
  IF ((Real_Gap > 10000.0) OR (Real_Gap < 1.0))
      OR (Hrec.Noise_Level > 60) OR (Hrec.Noise_Level < 0)
      OR (Hrec.Trail_Type = Bad_Data) THEN BEGIN
      {STUFFED_DATA!! - set trailarray[0] to 0 to avoid crash.}
      Trail_Array[0]:= 0;
      Gotoxy(2,22);
      WRITE('Trail ',Hrec.Trail_Num,' was ill.');
      WRITE(' - ',Hrec.Num_Samples,' Samples,  Noise = ',Hrec.Noise_Level-140)
  END
  ELSE
      IF (Hrec.Trail_Type IN Types_Wanted) THEN
         IF (   ((NOT Match_Only) AND (NOT No_Matches))
             OR (Match_Only AND (Hrec.Matched_With <> -1))
             OR (No_Matches AND (Hrec.Matched_With = -1))   ) THEN BEGIN
            Num_Trails_Considered:= Num_Trails_Considered + 1.0;
            Sample_Gap:= TRUNC (Real_Gap);
            Longval:= TRUNC(Hrec.Sample_Range[0]);
            SEEK (Sample_File, Longval);
            FOR Sample := 1 TO Sample_Gap DO
             BEGIN
                READ (Sample_File, Sample_Value);
                Trail_Array [Sample] :=
                     Sample_Value - 140 + Trail_Dbm_Alteration
             END;
            Trail_Array [0] := Sample_Gap
         END
         ELSE Trail_Array[0]:= 0          {This type not wanted.}
     ELSE Trail_Array[0]:= 0
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Get_Comms_Info ( Iteration : INTEGER; Bit_Rate, Block_Size  : REAL;
                    VAR Number_Of_Blocks, Total_Prob_Over_All_Blocks : REAL);

{ This routine will return the number of blocks got over on a }
{ trail, and the total probability = total blocks transmitted }
{ because the probabilities are in the range 0 to 1.          }

VAR  Where_In_Trail, End_Of_Trail, Cut_Off_Dbm       : REAL;
     Last_Frame_Prob                                  : REAL;
     Trail_Pos                                       : INTEGER;
     Dipped_Under_Minimum                            : BOOLEAN;


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Update_Trail_Pos;

BEGIN
   Trail_Pos:= TRUNC( (Where_In_Trail/500.0));
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Find_Turn_On;
   VAR Cur_Sig_Noise : INTEGER;
             Not_On : BOOLEAN;

{ Find where the trail samples are high enough for the turn on }
{ specified by the user.                                       }

BEGIN
   Not_On:= TRUE;
   WHILE (Not_On AND (Trail_Pos <= Trail_Array[0])) DO BEGIN
      Cur_Sig_Noise:= Trail_Array[ Trail_Pos] - (Hrec.Noise_Level - 140);
      IF (Cur_Sig_Noise < Sig_Noise_On) THEN
           Trail_Pos:= Trail_Pos + 1
      ELSE Not_On:= FALSE;
   END;
   Where_In_Trail:= (Trail_Pos * 5.0) * 100.0;
   Where_In_Trail:= Where_In_Trail + Sync_Xsecs;
   Dipped_Under_Minimum:= FALSE
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
FUNCTION Another_Block_Feasible : BOOLEAN;
{If time for block and turn on found.}

BEGIN
  IF ((Where_In_Trail + Block_Size_Xsecs) > End_Of_Trail) THEN
       Another_Block_Feasible:= FALSE
  ELSE Another_Block_Feasible:= TRUE;
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Init_Sample_Structures;

BEGIN
   End_Of_Trail:= ((Hrec.Num_Samples + 1.0) * 5.0) * 100.0 - 1.0; {In XSecs.}
   Where_In_Trail:= 500.0;
   Trail_Pos:= 1;
   Dipped_Under_Minimum:= FALSE;
   trail_array [hrec.num_samples + 1] := trail_array [hrec.num_samples]
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Prop_Delay;

BEGIN
   Where_In_Trail:= Where_In_Trail + Propagation_Delay;
   Update_Trail_Pos
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Receive_Probe;

BEGIN
   Where_In_Trail:= Where_In_Trail + Probe_Time;
   Update_Trail_Pos;
END;
{-----------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Switch_From_Receive_Wait;

BEGIN
   Where_In_Trail:= Where_In_Trail + Switch_From_Receive_Xsecs;
   Update_Trail_Pos
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Send_Sync;

BEGIN
  Where_In_Trail:= Where_In_Trail + Sync_Xsecs;
  Update_Trail_Pos
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Wait_Max_Of_2_Times_Prop_Delay_Or_Switch_From_Transmit;

BEGIN
   IF ((Propagation_Delay * 2.0) > Switch_From_Transmit_Xsecs) THEN
        Where_In_Trail:= Where_In_Trail + (Propagation_Delay * 2.0)
   ELSE Where_In_Trail:= Where_In_Trail + Switch_From_Transmit_Xsecs;
   Update_Trail_Pos
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Xsecs (Ipos : INTEGER) : REAL;

{ Returns how far we are (in time) into the trail. }

BEGIN
   Xsecs:= (Ipos * 5.0) * 100.0;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Going_Past_Next_Sample( Time_Still_Needed : REAL) : BOOLEAN;

BEGIN
  IF ((Time_Still_Needed + Where_In_Trail) >= Xsecs( Trail_Pos + 1)) THEN
       Going_Past_Next_Sample:= TRUE
  ELSE Going_Past_Next_Sample:= FALSE
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Real_Round (Reel  : REAL)  : REAL;

BEGIN
  Reel := Reel + 0.5;
  Real_Round := Reel - Frac (Reel)
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Send_Block( Send_Size_Xsecs : REAL; Ack_Involved : BOOLEAN);

{ The transmission of a block of data is performed by this   }
{ procedure. The size of the block in time is sent as a      }
{ parameter, as well as if it is data or an ACK being sent.  }
{ A data block only gets across the link if its ACK also     }
{ gets across.                                               }

    VAR Time_Still_Needed, Time_Taken, Ten_Pwr_Term, Num_Bits  : REAL;
                  Dbmpos, Bad_Prob, Sample_Power, Num_Dif_Samples : INTEGER;
                         A_Bit_Err_Prob, Cum_Prob, Area_Good_Prob : REAL;
                                                      Exp_Operand : REAL;
                                                         Curnoise : REAL;

        Bits_Array :  ARRAY[-140..-80] OF REAL;
            Anykey : CHAR;


BEGIN
   Fillchar( Bits_Array, Sizeof( Bits_Array), 0);
   Time_Still_Needed:= Send_Size_Xsecs;
   Curnoise:= Hrec.Noise_Level - 140.0;
   WHILE (Time_Still_Needed > 0.0) DO BEGIN
      Sample_Power:= Trail_Array[ Trail_Pos];
      Dipped_Under_Minimum:=
         (Sig_Noise_Off < (Sample_Power - Curnoise));
      IF (Going_Past_Next_Sample( Time_Still_Needed)) THEN BEGIN
          Time_Taken:= Xsecs(Trail_Pos + 1) - Where_In_Trail;
          Where_In_Trail:= Xsecs( Trail_Pos + 1);
          Trail_Pos:= Trail_Pos + 1;
      END
      ELSE BEGIN
           Time_Taken:= Time_Still_Needed;
           Where_In_Trail:= Where_In_Trail + Time_Still_Needed
      END;
      Time_Still_Needed:= Time_Still_Needed - Time_Taken;
      Num_Bits:= Bits_Per_Xsec * Time_Taken;
      IF (sample_power > -80.0) OR (Sample_power < -140.0) THEN
          IF debug THEN WRITELN('Sample_power ',sample_power,' !!!!!!');
      Bits_Array[ Sample_Power]:= Bits_Array[ Sample_Power] + Num_Bits;
   END;
   Cum_Prob:= 1.0;
   FOR Dbmpos:= -140 TO -80 DO
      IF (Bits_Array[ Dbmpos] <> 0.0) THEN BEGIN
         Ten_Pwr_Term:=
            Exp(((Dbmpos - Curnoise)/10.0) * Ln_10);
         Exp_Operand := 0.0 - ((Bandwidth * Ten_Pwr_Term)/Bit_Rate);
         IF Exp_Operand < Exp_Cutoff THEN
               A_Bit_Err_Prob:= 0.0
         ELSE A_Bit_Err_Prob := 0.5 * Exp( Exp_Operand);
         IF debug THEN
          BEGIN
            WRITE('bit err prob is ',a_bit_err_prob:20:15);
            WRITELN ('bits = ',bits_array [dbmpos], ' dbm = ', dbmpos)
          END;
          Exp_Operand:= Bits_Array[ Dbmpos] * Ln( 1.0 - A_Bit_Err_Prob);
          IF (Exp_Operand < Exp_Cutoff) THEN
                Area_Good_Prob:= 0.0
          ELSE Area_Good_Prob:= Exp(Exp_Operand);
           {1-Probability of error in bit powered to number of bits.}
         Cum_Prob:= Cum_Prob * Area_Good_Prob;
      END;
   IF (Ack_Involved) THEN BEGIN
      Cum_Prob:= Cum_Prob * Last_Frame_Prob;
      Total_Prob_Over_All_Blocks:= Total_Prob_Over_All_Blocks + Cum_Prob;
      Number_Of_Blocks:= Number_Of_Blocks + 1.0;
   END
   ELSE Last_Frame_Prob:= Cum_Prob
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Get_Info_From_Samples;

{ This procedure simulates the transmission of data across a trail. }
{ The protocol used can easily be seen from the way the procedure   }
{ is phrased.                                                       }


   CONST Frame = FALSE;
         Ack = TRUE;

    VAR Oldwhere : REAL;

BEGIN
   Init_Sample_Structures;
   Find_Turn_On;
   WHILE Another_Block_Feasible DO BEGIN
       Receive_Probe;
       Oldwhere:= Where_In_Trail;
       Switch_From_Receive_Wait;
       Send_Sync;
       WHILE ((NOT Dipped_Under_Minimum) AND
              Another_Block_Feasible) DO BEGIN
           Send_Block( Block_Size_Xsecs, Frame);
           Wait_Max_Of_2_Times_Prop_Delay_Or_Switch_From_Transmit;
           Trans_Xsecs:= Trans_Xsecs + (Where_In_Trail - Oldwhere);
           Send_Block( Ack_Size_Xsecs, Ack);
           Oldwhere:= Where_In_Trail;
           Switch_From_Receive_Wait
       END;
       Trans_Xsecs:= Trans_Xsecs + (Where_In_Trail - Oldwhere);
       Find_Turn_On;
   END;
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
BEGIN
  Number_Of_Blocks:= 0.0;
  Total_Prob_Over_All_Blocks:= 0.0;
  Get_Info_From_Samples;
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Compute_Info_For_All_Files (Start_File, Stop_File  : INTEGER;
          VAR  Best_Bit, Best_Block, Number_Bits_Accross     : REAL);

{ The link used is specified by the files that are used.  This }
{ procedure will go through all the user selected files       }
{ computing the throughput.                                   }

VAR  Iteration, Bit_Index, Block_Index                  : INTEGER;
     File_Number                                        : INTEGER;
     Block_Size, Block_Start, Block_Stop                : REAL;
     Bit_Rate, Bit_Start, Bit_Stop                      : REAL;
     Bit_Scale, Block_Scale, Db_Scale                   : REAL;
     Number_Of_Blocks, Total_Good                       : REAL;
     Max_Data_Sent                                      : REAL;
     Files_Availible                                    : BOOLEAN;
     It_Count                                           : INTEGER;
     Bit_Top, Block_Top                                 : INTEGER;
```

```
{-----------------------------------------------------------------------}
PROCEDURE  Get_New_Starts_And_Stops;
    VAR Data_Bits_Sent : REAL;
        Bit_Index, Block_Index : LONGINT;
{ Compute the new breadth of the search. }

BEGIN
  Max_Data_Sent := -1.0;
  FOR Bit_Index := 1 TO Bit_Index_Range DO
     FOR Block_Index := 1 TO Block_Index_Range DO
       BEGIN
          Data_Bits_Sent:= (Block_Value[ Iteration, Block_Index] - Crc_Length)
                      * Comms_Info [ Iteration, Bit_Index, Block_Index, 2];
          IF Data_Bits_Sent > Max_Data_Sent THEN
           BEGIN
              Max_Data_Sent := Data_Bits_Sent;
              Best_Bit := Bit_Value[ Iteration, Bit_Index];
              Best_Block := Block_Value[ Iteration, Block_Index];
           END;
        END;
  Block_Start := Best_Block - Block_Scale / 2;
  Block_Stop  := Best_Block + Block_Scale / 2;
  Bit_Start   := Best_Bit - Bit_Scale / 2;
  Bit_Stop    := Best_Bit + Bit_Scale / 2
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
PROCEDURE Get_Db_Turn_On_Off_Level( Bitrate : REAL);
  VAR Log10term : REAL;

BEGIN
   Log10term:= 0.0 - (Bitrate/Bandwidth * Ln( 2.0 * On_Bit_Err_Rate));
   Sig_Noise_On:= (Ln( Log10term) / Ln_10) * 10.0;
   IF Debug THEN WRITE('LEVEL ON = ', Sig_Noise_On:20:20);
   Log10term:= 0.0 - (Bitrate/Bandwidth * Ln( 2.0 * Off_Bit_Err_Rate));
   Sig_Noise_Off:= (Ln( Log10term) / Ln_10) * 10.0;
   IF Debug THEN WRITELN('LEVEL OFF = ',Sig_Noise_Off);
END;
{-----------------------------------------------------------------------}



{-----------------------------------------------------------------------}
PROCEDURE Fill_Type_Counts( Type_In : BYTE; Num_Good_Blocks : REAL);
   VAR Pos : BYTE;

BEGIN
   Typecount[ Type_In]:= Typecount[ Type_In] + Num_Good_Blocks;
END;
{-----------------------------------------------------------------------}



{-----------------------------------------------------------------------}
BEGIN
  Menu_Head;
  Block_Start := Start_Block;
  Block_Stop  := Stop_Block;
  Bit_Start   := Start_Bit;
  Bit_Stop    := Stop_Bit;
  Fillchar (Comms_Info, Sizeof (Comms_Info), 0);
  IF Optimising THEN
       It_Count:= Number_Of_Iterations
  ELSE It_Count:= 1;
  FOR Iteration := 1 TO It_Count DO
   BEGIN
     IF Optimising THEN BEGIN
        Gotoxy (5, 5);
        WRITE ('Current search depth is ', Iteration);
        Clreol;
        Bit_Scale := (Bit_Stop - Bit_Start) / Float (Bit_Index_Range);
        Block_Scale := (Block_Stop - Block_Start) / Block_Index_Range;
     END
```

```
      ELSE BEGIN
         Bit_Rate:= User_Bit_Rate;
         Block_Size:= User_Block_Size;
      END;
      FOR File_Number := Start_File TO Stop_File DO
       BEGIN
         Ready_Files (File_Number, Files_Availible);
         IF Files_Availible THEN
          BEGIN
            Gotoxy (5, 7);
            WRITE ('Computing bit rate for file ', File_Table [File_Number]);
            Clreol;
            Gotoxy (5, 9);
            WRITE ('Processing trail ');
            Clreol;
            WHILE NOT (EOF (Header_File)) DO
            BEGIN
              REPEAT
                 Get_Next_Trail;
              UNTIL ((Trail_Array[0] <> 0) OR (EOF( Header_File)));
              IF (Trail_Array[0] <> 0) THEN BEGIN
               Gotoxy (22, 9);
               WRITE (Hrec.Trail_Num);
               IF Optimising THEN BEGIN
                  FOR Bit_Index := 1 TO Bit_Index_Range DO
                     Bit_Value [ Iteration, Bit_Index] :=
                        Bit_Start + Bit_Index * Bit_Scale - Bit_Scale / 2;
                  FOR Block_Index := 1 TO Block_Index_Range DO
                    Block_Value [ Iteration, Block_Index] :=
                     Block_Start + Block_Index * Block_Scale - Block_Scale / 2;
                  Bit_Top:= Bit_Index_Range;
                  Block_Top:= Block_Index_Range;
               END
               ELSE BEGIN
                     Bit_Top:= 1;
                     Block_Top:= 1;
               END;
               FOR Bit_Index := 1 TO Bit_Top DO
                  FOR Block_Index := 1 TO Block_Top DO
                    BEGIN
                       IF Optimising THEN BEGIN
                          Bit_Rate := Bit_Value [ Iteration, Bit_Index];
                          Block_Size := Block_Value [ Iteration, Block_Index];
                       END;
                       Get_Db_Turn_On_Off_Level( Bit_Rate);
                       Data_Bits_Per_Block := Block_Size - Crc_Length;
                       Ack_Size_Xsecs := Ack_Length / Bit_Rate * 100000.0;
                       Block_Size_Xsecs := Block_Size / Bit_Rate * 100000.0;
                       Probe_Time := Probe_Length / Bit_Rate * 100000.0;
                       Bits_Per_Xsec := Bit_Rate / 100000.0;
                       Get_Comms_Info ( Iteration, Bit_Rate, Block_Size,
                                        Number_Of_Blocks, Total_Good);
Total_Good:4:4);        IF  debug  THEN  WRITELN  (Number_Of_Blocks:1:0,  '  ',
                       Comms_Info [ Iteration, Bit_Index, Block_Index, 1] :=
                          Comms_Info [ Iteration, Bit_Index, Block_Index, 1] +
                                Number_Of_Blocks;
                       Comms_Info [ Iteration, Bit_Index, Block_Index, 2] :=
                          Comms_Info [ Iteration, Bit_Index, Block_Index, 2] +
                            Total_Good;
                       Fill_Type_Counts( Hrec.Trail_Type, Total_Good);
                    END                              {Index loops.  }
              END                                    {Trail OK IF.  }
            END;                                     {While file.   }
            Close_Files;
         END;                                        {File here IF. }
       END;                                          {All files loop}
      IF Optimising THEN
         Get_New_Starts_And_Stops;
    END;                                        {Iteration loop}
   IF (NOT Optimising) THEN
      Max_Data_Sent:= (User_Block_Size - Crc_Length) * Comms_Info[ 1,1,1,2];
   Number_Bits_Accross := Max_Data_Sent {/ 8.0}
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Iteration_Head( Iteration : INTEGER);

BEGIN
   WRITE( Outf, CHR(12));
   WRITELN(Outf,'  Datacom Optimisation Process   -    Iteration ',Iteration);
   WRITELN(Outf,'  ------------------------------------------------');
   WRITELN(Outf);
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Show_Parameters_Used;

BEGIN
  WRITELN(Outf);
  WRITELN(Outf);
  WRITELN(Outf,'                    Parameters Used.');
  WRITELN(Outf,'                    ++++++++++++++++');
  WRITELN(Outf);
  WRITELN(Outf,'Path length (kilometres)    :  ', Path_Length:10:0);
  WRITELN(Outf,'Bandwidth (Hertz)           :  ', Bandwidth:10);
  WRITELN(Outf,'Probe length (bits)         :  ', Probe_Length:10);
  WRITELN(Outf,'CRC length (bits)           :  ', Crc_Length:10);
  WRITELN(Outf,'ACK length (bits)           :  ', Ack_Length:10);
  WRITELN(Outf,'Modem sync time (MilliSecs) :  ', (Sync_Xsecs/100.0):10:0);
  WRITELN(Outf,'Turn on error rate          :  ',On_Bit_Err_Rate:10:5);
  WRITELN(Outf,'Turn off error rate         :  ',Off_Bit_Err_Rate:10:5);
  WRITELN(Outf,'Transmit power (Watts)      :  ',Transmit_Power:10);
  WRITELN(Outf,'Measured power (Watts)      :  ',Measured_Power:10);
  WRITELN(Outf,'dBm alteration on wattage   :  ',Trail_Dbm_Alteration:10);
  WRITELN(Outf,'Minimum block size (bits)   :  ',Start_Block:10:0);
  WRITELN(Outf,'Maximum block size (bits)   :  ',Stop_Block:10:0);
  WRITELN(Outf,'Minimum bit rate (Kbps)     :  ',(Start_Bit/1024.0):10:2);
  WRITELN(Outf,'Maximum bit rate (Kbps)     :  ',(Stop_Bit/1024.0):10:2);
  WRITE(Outf,'Remote turnaround time (transmit to receive) MSecs   : ');
  WRITELN(Outf,(Switch_From_Transmit_Xsecs/100.0):1:0);
  WRITE(Outf,'Remote turnaround time (receive to transmit) MSecs   : ');
  WRITELN(Outf, (Switch_From_Receive_Xsecs/100.0):1:0);
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Show_Single_Result(   Number_Bits_Across : REAL;
                              Start_File, Stop_File : INTEGER);

VAR       Num_Hours : INTEGER;
        Trans_Joules : REAL;
     Ublocks, Uprob : REAL;
```

```
BEGIN
   Ublocks:= Comms_Info [ 1,1,1,1];
   Uprob:= Comms_Info [ 1,1,1,2];
   IF (Sink <> Printer) THEN BEGIN
        Menu_Head;
        Gotoxy (1, 5);
   END
   ELSE BEGIN
      WRITE(Outf,CHR(12));
      WRITELN(Outf,'            Datacom Result ');
      WRITELN(Outf,'            -------------- ');
      WRITE(Outf,'       considering ',Tystr,' types of trails');
      WRITELN(Outf, Match_Str);
      WRITE(Outf,'       There were ',Num_Trails_Considered:1:0);
      WRITELN(Outf,' trails considered.');
      WRITELN(Outf);
   END;
   Num_Hours:= (Stop_File - Start_File + 1);
   WRITELN (Outf,'     Bit-rate used    : ', User_Bit_Rate :1:0);
   WRITELN (Outf,'     Block size used  : ', User_Block_Size :1:0);
   WRITELN( Outf);
   WRITE   (Outf, '     Number of data bits transferred in ', Num_Hours);
   WRITELN (Outf,' hours was  : ', Number_Bits_Across :1:0);
   WRITE   (Outf, '     This gives a rate of ');
   WRITE(Outf, (Number_Bits_Across/(3600.0 * Num_Hours)):1:0);
   WRITELN(Outf,' correct data bits per second.');
   WRITELN(Outf);
   WRITE(Outf,'Blocks tried : ');
   WRITE(Outf,Ublocks:1:0);
   WRITE(Outf,'     Blocks correct : ');
   IF Ublocks = 0.0 THEN
       WRITE (Outf,0.00 :1:0)
   ELSE WRITE (Outf,Uprob:1:0);
   WRITE(Outf,'     % blocks correct : ');
   IF Ublocks = 0.0 THEN
       WRITE (Outf,0.00 :7:3)
   ELSE
       WRITE (Outf, (Uprob/Ublocks) * 100.0 :7:3);
   WRITELN(Outf,' %');
   WRITE(Outf,'Non-data bits (ie. control bits) percentage : ');
   WRITELN(Outf, ((Crc_Length/User_Block_Size) * 100.0):14:6,' %');
   Trans_Joules:= (Trans_Xsecs/100000.0) * Transmit_Power; {Secs * power}
   WRITE(Outf,'Time remote transmitting was ');
   WRITE(Outf, (Trans_Xsecs/100000.0):7:3, ' seconds,');
   WRITELN(Outf,' using ',Trans_Joules:1:0,' Joules energy.');
   IF Sink = Printer THEN
       Show_Parameters_Used
   ELSE BEGIN
       Gotoxy(1,24);
       WRITE ('   Hit any key to continue  ');
       anykey:= Readkey;
       Clrscr;
    END;
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Show_Results( Number_Bits_Across,
                                Best_Block_Size, Best_Bit_Rate : REAL;
                  Start_File, Stop_File, Bit_Index, Block_Index : INTEGER);

VAR Iteration, Num_Hours : INTEGER;

BEGIN
  IF (NOT Optimising) THEN
       Show_Single_Result (Number_Bits_Across, Start_File, Stop_File)
  ELSE BEGIN
   Num_Trails_Considered:= Num_Trails_Considered/Number_Of_Iterations;
   IF (Sink <> Printer) THEN BEGIN
        Menu_Head;
        Gotoxy (1, 3);
        WRITE(Outf,'       Considering ',Tystr,' types of trails');
        WRITELN(Outf, Match_Str);
        WRITE(Outf,'       There were ',Num_Trails_Considered:1:0);
```

```
      WRITELN(Outf,' trails considered.');
   END
   ELSE BEGIN
      WRITE(Outf,CHR(12));
      WRITE(Outf,'   Datacom Optimisation Result after ');
      WRITELN(Outf,Number_Of_Iterations,' Iterations.');
      WRITELN(Outf,'   ------------------------------------------');
      WRITE(Outf,'       considering ',Tystr,' types of trails');
      WRITELN(Outf, Match_Str);
      WRITE(Outf,'       There were ',Num_Trails_Considered:1:0);
      WRITELN(Outf,' trails considered.');
      WRITELN(Outf);
   END;
   Num_Hours:= (Stop_File - Start_File + 1);
   WRITELN (Outf,'     Best bit-rate is    : ', Best_Bit_Rate :1:0);
   WRITELN (Outf,'     Best block size is  : ', Best_Block_Size :1:0);
   WRITELN( Outf);
   WRITE    (Outf, '     Number of data bits transferred in ', Num_Hours);
   WRITELN (Outf,' hours was  : ', Number_Bits_Across :1:0);
   WRITE(Outf,     '     This gives a rate of ');
   WRITE(Outf, (Number_Bits_Across/(3600.0 * Num_Hours)):1:0);
   WRITELN(Outf,' correct data bits per second.');
   Show_Parameters_Used;
   IF (Sink <> Printer) THEN BEGIN
       Gotoxy(1,24);
       WRITE ('   Hit any key to continue  ');
       anykey:= Readkey;
       Clrscr;
    END;
    FOR Iteration:= 1 TO Number_Of_Iterations DO BEGIN;
      IF Sink = Printer THEN
         Iteration_Head( Iteration);
      WRITE(Outf,'Bit Rate  Block Size  Blocks tried   Block % OK  ');
      WRITELN(Outf,'Control Bit %  Data Bits over');
      WRITE(Outf,'~~~~~~~~  ~~~~~~~~~~  ~~~~~~~~~~~~   ~~~~~~~~~~  ');
      WRITELN(Outf,'~~~~~~~~~~~~  ~~~~~~~~~~~~~');
      FOR Bit_Index := 1 TO Bit_Index_Range DO
        FOR Block_Index := 1 TO Block_Index_Range DO
          BEGIN
             WRITE(Outf,Bit_Value [ Iteration, Bit_Index]:8:0);
             WRITE(Outf,Block_Value [ Iteration, Block_Index] :12:0);
             WRITE(Outf,Comms_Info [ Iteration, Bit_Index, Block_Index, 1]:14:0);
             IF Comms_Info [ Iteration, Bit_Index,Block_Index,1] = 0 THEN
                WRITE (Outf,0.00 :13:3)
             ELSE
                WRITE (Outf,Comms_Info [ Iteration, Bit_Index, Block_Index, 2] /
                       Comms_Info [ Iteration, Bit_Index, Block_Index, 1]
                                  * 100.0 :13:3);
             WRITE(Outf, ((Crc_Length/Block_Value[ Iteration, Block_Index])
                                                    * 100.0):14:6);
             WRITE (Outf,Comms_Info [ Iteration, Bit_Index, Block_Index, 2] *
               (Block_Value[ Iteration, Block_Index] - Crc_Length) :14:0);
             IF (Bit_Value [ Iteration, Bit_Index] = Best_Bit_Rate) AND
                (Block_Value [ Iteration, Block_Index] = Best_Block_Size)
                AND (Comms_Info[ Iteration, Bit_Index, Block_Index, 2]
                                                    > 0.0) THEN
                          WRITELN (Outf,' *')
             ELSE         WRITELN(Outf)
          END
    END;
  END
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Auto_Process_Files_In_Directory;

VAR  Junk                                                   : String;
     Start_File, Stop_File                                  : INTEGER;
     Number_Bits_Across, Best_Block_Size, Best_Bit_Rate     : REAL;
     Bit_Index, Block_Index                                 : INTEGER;

BEGIN
  Chdir( Path);
  Fill_File_Table;
  IF Number_Files_In_Table = 0 THEN
     Junk := Users_Response_To (1, 'No meteor data files found (OK)?')
  ELSE
   BEGIN
     List_File_Table_To_Screen;
     Start_File := Get_User_Int ('Enter start file number :',
        1, Number_Files_In_Table);
     Stop_File := Get_User_Int ('Enter stop file number :',
        1, Number_Files_In_Table);
     Num_Trails_Considered:= 0.0;
     Compute_Info_For_All_Files (Start_File, Stop_File,
        Best_Bit_Rate, Best_Block_Size, Number_Bits_Across);
     Chdir( Olddir);
     IF Start_File <= Stop_File THEN
         Show_Results( Number_Bits_Across,Best_Block_Size, Best_Bit_Rate,
                          Start_File, Stop_File, Bit_Index, Block_Index);
   END
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Handle_Initial_Parameters;
   VAR Switch_From_Transmit_Msec, Switch_From_Receive_Msec : REAL;
                                           On_Rate, Off_Rate : INTEGER;
                                                 Sync_Msecs : REAL;
                                                    Newpath : String;
{ This will allow the user to use the default parameter settings }
{ or change them.                                                }


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Set_Default_Parameters;

BEGIN
   Path_Length:= 2000.0;
   Bandwidth:= 2000;
   Probe_Length:= 10;
   Crc_Length:= 8;
   Ack_Length:= 8;
   Sync_Msecs:= 2.0;
   On_Rate:= 3;                  {On if one error in 1000 bits.}
   Off_Rate:= 2;                 {Off if one error in 100 bits.}
   Switch_From_Transmit_Msec:= 1.0;
   Switch_From_Receive_Msec:= 2.0;
   Transmit_Power:= 400;
   Measured_Power:= 400;
   Start_Block:= 80.0;
   Stop_Block:= 19200.0;
   Start_Bit:= 256.0;
   Stop_Bit:= 102400.0;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Get_User_Parameters;

BEGIN
  Menu_Head;
  Gotoxy (1, 6);
  WRITELN('                    USER  PARAMETER  ENTRY');
  WRITELN('                    ======================');
  WRITELN;
  WRITE ('Enter path-length in Km.      :  ');
  READLN (Path_Length);
  WRITE ('Enter bandwidth in Hz         :  ');
  READLN (Bandwidth);
  WRITE ('Enter probe length in bits    :  ');
  READLN (Probe_Length);
  WRITE ('Enter CRC length in bits      :  ');
  READLN (Crc_Length);
  WRITE ('Enter ACK length in bits      :  ');
  READLN (Ack_Length);
  WRITE ('Enter modem sync time in Ms.  :  ');
  READLN (Sync_Msecs);
  WRITE('Enter transmit power  (watts) :  ');
  READLN( Transmit_Power);
  WRITE('Enter measured power  (watts) :  ');
  READLN( Measured_Power);
  WRITE('Remote turnaround time (transmit to receive) MSecs    : ');
  READLN( Switch_From_Transmit_Msec);
  WRITE('Remote turnaround time (receive to transmit) MSecs    : ');
  READLN( Switch_From_Receive_Msec);
  WRITE('Turn on error rate (1 in (10 to power X), enter X)    : ');
  READLN( On_Rate);
  WRITE('Turn off error rate (1 in (10 to power X), enter X)   : ');
  READLN( Off_Rate);
  WRITE('Enter minimum block size (bits) :  ');
  READLN( Start_Block);
  WRITE('Enter maximum block size (bits) :  ');
  READLN( Stop_Block);
  WRITE('Enter minimum bit rate (bps)    :  ');
  READLN( Start_Bit);
  WRITE('Enter maximum bit rate (bps)    :  ');
  READLN( Stop_Bit);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Ten_Power( Rate : INTEGER) : REAL;
   VAR Tot : REAL;
         I : INTEGER;

BEGIN
  Tot:= 1.0;
  FOR I:= 1 TO Rate DO
      Tot:= Tot * 10.0;
  Ten_Power:= Tot
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Set_Vars_On_Parameters;

BEGIN
  Propagation_Delay := Path_Length / Speed_Of_Light;
  Sync_Xsecs := Sync_Msecs * 100.0;
  Switch_From_Transmit_Xsecs:= Switch_From_Transmit_Msec * 100.0;
  Switch_From_Receive_Xsecs:= Switch_From_Receive_Msec * 100.0;
  On_Bit_Err_Rate:= 1.0/Ten_Power( On_Rate);
  Off_Bit_Err_Rate:= 1.0/Ten_Power( Off_Rate);
  Trail_Dbm_Alteration:=
       ROUND(10.0 * Ln( Transmit_Power/Measured_Power)/Ln_10);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Show_Default_Parameters;

BEGIN
  Menu_Head;
  WRITELN;
  WRITELN('                  Default  Parameters.');
  WRITELN('                  ===================');
  WRITELN;
  WRITELN('Path length (kilometres)     :  ', Path_Length:10:0);
  WRITELN('Bandwidth (Hertz)            :  ', Bandwidth:10);
  WRITELN('Probe length (bits)          :  ', Probe_Length:10);
  WRITELN('CRC length (bits)            :  ', Crc_Length:10);
  WRITELN('ACK length (bits)            :  ', Ack_Length:10);
  WRITELN('Modem sync time (MilliSecs)  :  ', Sync_Msecs:10:0);
  WRITELN('Turn on error rate           :  ',(1.0/Ten_Power(On_Rate)):10:5);
  WRITELN('Turn off error rate          :  ',(1.0/Ten_Power(Off_Rate)):10:5);
  WRITELN('Transmit power (watts)       :  ',Transmit_Power:10);
  WRITELN('Receive power (watts)        :  ',Measured_Power:10);
  WRITELN('Minimum block size (bits)    :  ',Start_Block:10:0);
  WRITELN('Maximum block size (bits)    :  ',Stop_Block:10:0);
  WRITELN('Minimum bit rate (Kbps)      :  ',(Start_Bit/1024.0):10:2);
  WRITELN('Maximum bit rate (Kbps)      :  ',(Stop_Bit/1024.0):10:2);
  WRITE('Remote turnaround time (transmit to receive) MSecs   : ');
  WRITELN( Switch_From_Transmit_Msec:1:0);
  WRITE('Remote turnaround time (receive to transmit) MSecs   : ');
  WRITELN( Switch_From_Receive_Msec:1:0);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Determine_And_Set_Sink;
   VAR Sinkchar : CHAR;

BEGIN
  Gotoxy(1,24);
  WRITE('Send output to Console or to Printer (C/P) ? ');
  REPEAT
     Sinkchar:= Upcase( Readkey);
  UNTIL (Sinkchar IN ['C','P']);
  IF (Sinkchar = 'C') THEN BEGIN
      ASSIGN( Outf, '');
      RESET( Outf);
      Sink:= Console;
  END
  ELSE BEGIN
      ASSIGN( Outf, 'PRINTER');
      REWRITE( Outf);
      Sink:= Printer
  END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Ok_Defaults_Or_Get_User_Entries;
  VAR Ok_Key : CHAR;

BEGIN
   Show_Default_Parameters;
   Gotoxy(1,24);
   WRITE('Use these parameters? (Y/N)    ');
   Ok_Key:= Readkey;
   Ok_Key:= Upcase( Ok_Key);
   IF (Ok_Key <> 'Y') THEN
       Get_User_Parameters;
   Determine_And_Set_Sink;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{--------------------------------------------------------------------}
BEGIN
  Path:= 'C:\WORK\METEOR\DATA';
  Getdir( 0, Olddir);
  REPEAT
    WRITE('Please enter path (default is \work\meteor\data) : ');
    READLN( Newpath);
    IF (Newpath = '') THEN
       Newpath:= Path;
    {$I-} Chdir( Newpath); {$I+}
  UNTIL (Ioresult = 0);
  Path:= Newpath;
  Chdir( Olddir);
  Set_Default_Parameters;
  Ok_Defaults_Or_Get_User_Entries;
  Set_Vars_On_Parameters;
  Bells:= [6,15];
  U_Dense:= [5,9,10,11,12,13,14,16,17,18,24,29];
  O_Dense:= [19, 20, 21, 26, 27, 28];
  Mush:=  [3, 7, 8, 23, 25];
  Gothic:= [22];
  Fillchar( Typecount, Sizeof(Typecount), 0);
END;
{--------------------------------------------------------------------}


{--------------------------------------------------------------------}
PROCEDURE Determine_Whether_To_Optimise_Or_Not;
    VAR Opchoice : CHAR;

BEGIN
   Menu_Head;
   Gotoxy(1,10);
   WRITELN('You have two options here - if you wish to have the program');
   WRITELN('optimise bit rate and block size then press "O", if instead');
   WRITELN('you desire to enter rate and size yourself enter "U".');
   WRITELN;
   WRITE('Please enter choice (O/U) : ');
   REPEAT
      Opchoice:= Upcase( Readkey);
   UNTIL (Opchoice IN ['O','U']);
   WRITELN( Opchoice);
   Optimising:= (Opchoice = 'O');
   IF (NOT Optimising) THEN BEGIN
       WRITE('Please enter block size in bits : ');
       READLN( User_Block_Size);
       WRITE('Please enter bit rate in bits per second : ');
       READLN( User_Bit_Rate);
       Remote_Time_Wanted:= TRUE;
       Trans_Xsecs:= 0.0;
   END
END;
{--------------------------------------------------------------------}



{--------------------------------------------------------------------}
PROCEDURE Get_Types_Wanted;
   VAR Match_Choice, Type_Opt : CHAR;
                     Num, Err : INTEGER;
                       Numstr : Str12;

BEGIN
   Menu_Head;
   WRITELN;
   WRITELN('          TYPE  SELECTION.');
   WRITELN;
   Menu_Write('A =  All types.');
   Menu_Write('B =  Bell type.');
   Menu_Write('C =  Classic underdense type.');
   Menu_Write('M =  Mush type.');
   Menu_Write('O =  Overdense type.');
   Menu_Write('S =  Select one of original action types (2-29).');
   Menu_Write('W =  Weird type.');
   Gotoxy( 1,24);
```

```
  WRITE('Please enter option : ');
  Type_Opt:= Get_Valid_Choice_From( ['A','B','C','M','O','S','W']);
  IF (Type_Opt = 'S') THEN BEGIN
      Gotoxy(1,24);
      Clreol;
      Gotoxy(1,24);
      WRITE('Please enter type number : ');
      REPEAT
         Gotoxy(28,24);
         READLN( Numstr);
         VAL( Numstr, Num, Err);
         IF (Num < 2) OR (Num > 29) THEN
             Err:= 1;
      UNTIL (Err = 0);
      Types_Wanted:= [Num];
      Tystr:= 'Type ' + Numstr;
      Gotoxy(1,24);
      Clreol
  END;
  Gotoxy(1,24);
  WRITE('All trails, Matched trails, or Non-matched trails (A/M/N) ? ');
  Match_Choice:= Get_Valid_Choice_From( ['A','M','N']);
  Match_Only:= (Match_Choice = 'M');
  No_Matches:= (Match_Choice = 'N');
  IF Match_Only THEN
       Match_Str:= ' - matched trails only.'
  ELSE
       IF No_Matches THEN
            Match_Str:= ' - non-matched trails only.'
       ELSE Match_Str:= '.';
  CASE Type_Opt OF
      'A' : BEGIN
               Types_Wanted:= [2..29];
               Tystr:= 'All';
            END;
      'B' : BEGIN
               Types_Wanted:= Bells;
               Tystr:= 'Bell'
            END;
      'C' : BEGIN
               Types_Wanted:= U_Dense;
               Tystr:= 'Underdense'
            END;
      'M' : BEGIN
               Types_Wanted:= Mush;
               Tystr:= 'Mush';
            END;
      'O' : BEGIN
               Types_Wanted:= O_Dense;
               Tystr:= 'Overdense';
            END;
      'W' : BEGIN
               Tystr:= 'Weird';
               Types_Wanted:= Gothic
            END
    END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Write_Type_Results;
  VAR Typeres : FILE OF Typearr;

BEGIN
   ASSIGN( Typeres, 'TYPERES');
   REWRITE( Typeres);
   WRITE( Typeres, Typecount);
   CLOSE( Typeres);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
BEGIN                              {Main Program}
  Handle_Initial_Parameters;
  Get_Types_Wanted;
  Determine_Whether_To_Optimise_Or_Not;
  Auto_Process_Files_In_Directory;
  Write_Type_Results;
  IF (Sink = Printer) THEN
     CLOSE( Outf);
  Chdir( Olddir);
END.
```

Appendix L - Source Listing of the NewWait Program

(Wait time preprocessor)

```
{$N+}     {8087 on board}
{$R-}     {Range checking off}
{$B+}     {Boolean complete evaluation on}
{$S+}     {Stack checking on}
{$I+}     {I/O checking on}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}

PROGRAM Get_Wait_Rates;
{Preprocessor for wait time analysis -  by  Stuart Melville.}


USES
  Crt,
  Dos,
  Library;


CONST  File_Table_Size       = 100;
       Modelled_Trail_Types  : SET OF BYTE = [];


       Number_Of_Iterations = 3;     { Search depth    }
       Bit_Index_Range      = 4;     {     "      "    }
       Block_Index_Range    = 4;     {     "      "    }

       Speed_Of_Light       = 300000.0;

       Max_Iter             = 5;     {Max iterations.}
       debug                = FALSE;
       Ln_10                = 2.3025851;
       Exp_Cutoff           = -88.0;
       Max_Sig_Noise_Gap    = 42;    {dBm.}

TYPE   Unsigned = ARRAY [1..2] OF BYTE;

       Str80          = String [80];
       Str40          = String [40];
       Str20          = String [20];
       Str14          = String [14];
       Str12          = String [12];
       Str10          = String [10];
       Str5           = String [5];
       Str2           = String [2];

       Time_Rec = RECORD
          Hours, Mins, Secs, Huns  : BYTE
       END;

       Dbs_Over_Noise = (Plus6, Plus9, Plus12, Plus15);

       Trail_Header = RECORD
          Trail_Num             : INTEGER;
          Start_Time            : Time_Rec;
          Elapsed_Time          : REAL;
          Num_Samples           : INTEGER;
          Sample_Range          : ARRAY [0..1] OF REAL;
          Noise_Level           : INTEGER;
          Number_Of_Fades       : INTEGER;
          Peak_Time             : INTEGER;
          Peak_Strength         : INTEGER;
          Signal_Present_Time   : ARRAY [0..1] OF Unsigned;
          Useful_Time           : ARRAY [Plus6..Plus15] OF Unsigned;
          Trail_Type            : BYTE;
          Sample_Mean           : REAL;
          Matched_With          : INTEGER;
          Spare_1, Spare_2      : INTEGER;
          Errors_Found          : BYTE
       END;
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
      Header_Fil    = FILE OF Trail_Header;
      Sample_Fil    = FILE OF BYTE;
      Rate_Arr      = ARRAY[1..400] OF REAL;
      Rate_Array_File = FILE OF Rate_Arr;
      Data_Array    = ARRAY [0..10000] OF INTEGER;
      Numset        = SET OF 2..29;
      Charset       = SET OF CHAR;
      Typeset       = SET OF 1..30;

VAR   File_Table                          : ARRAY [1..File_Table_Size] OF Str12;
                  Newpath : String;
      Number_Files_In_Table               : INTEGER;
      Header_File                         : Header_Fil;
      Sample_File                         : Sample_Fil;
      Hrec                                : Trail_Header;
      Trail_Array                         : Data_Array;
      Probe_Length, Ack_Length            : INTEGER;
      Bandwidth, Crc_Length               : INTEGER;
      Data_Bits_Per_Block                 : REAL;
      Olddir, Path : String;
      D : INTEGER;
      Probe_Time, Ack_Size_Xsecs          : REAL;
      Propagation_Delay, Path_Length      : REAL;
      Block_Size_Xsecs                    : REAL;
      Bits_Per_Xsec, Sync_Xsecs           : REAL;
      Anykey                              : CHAR;
      Bit_Value                           : ARRAY [1..Max_Iter,1..Bit_Index_Range]
                                                                    OF REAL;
      Block_Value                         : ARRAY [1..Max_Iter,1..Block_Index_Range]
                                                                    OF REAL;
      Comms_Info                          : ARRAY [1..Max_Iter,1..Bit_Index_Range,
                                                   1..Block_Index_Range,
                                                   1..2] OF REAL;
      Sig_Noise_On, Sig_Noise_Off          : REAL;
      On_Bit_Err_Rate                      : REAL;
      Off_Bit_Err_Rate                     : REAL;
      Switch_From_Transmit_Xsecs           : REAL;
      Switch_From_Receive_Xsecs            : REAL;
      Outf                                 : TEXT;
      Types_Used                           : Typeset;
      Ratf                                 : Rate_Array_File;
      Sink                                 : BOOLEAN;
      Trail_Dbm_Alteration                 : INTEGER;
      Transmit_Power, Measured_Power       : INTEGER;
      User_Block_Size, User_Bit_Rate       : REAL;
      Optimising                           : BOOLEAN;
      Tystr                                : Str20;
      Types_Wanted                         : Numset;
      Match_Only, No_Matches               : BOOLEAN;
      Num_Trails_Considered                : REAL;
      Match_Str                            : Str40;
      Remote_Time_Wanted                   : BOOLEAN;
      Trans_Xsecs                          : REAL;
      N0, Eb_To_N0, Pr, Bitr               : REAL;
      Bitsum                               : REAL;
      File_Used : INTEGER;
      Akey : CHAR;
      Rate_Array, Bits_Array               : Rate_Arr;
      Sig_Noise_Table                      : ARRAY[1..Max_Sig_Noise_Gap] OF WORD;
      {Gives a look-up to allow dBm spread where necessary.}


{-------------------------------------------------------------------------}
PROCEDURE debugwrite( st : string);
 VAR akey : CHAR;
BEGIN
   WRITELN( st);
   akey:= readkey;
END;
{-------------------------------------------------------------------------}
```

```
{---------------------------------------------------------------------}
PROCEDURE Menu_Write( St_In : Str80);
{Writes a string in 'menu' fashion. Does this by highlighting first}
{letter of string, (the option character). eg. If passed the string}
{"A =  Abort program", it would print out a  highlighted "A",  then}
{the rest of the string at lower intensity.                        }

BEGIN
   Normvideo;
   WRITE(St_In[1]);
   St_In[1]:= ' ';
   Lowvideo;
   WRITELN(St_In);
   Normvideo;
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
FUNCTION Get_Valid_Choice_From( Ok_Chars : Charset) : CHAR;
   VAR Choice : CHAR;

BEGIN
   REPEAT
      Choice:= Upcase( Readkey)
   UNTIL (Choice IN Ok_Chars);
   Get_Valid_Choice_From:= Choice
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
FUNCTION Users_Response_To (Chars   : INTEGER; Question   : Str80) :Str12;

VAR  Answer       : Str12;
     Pos          : BYTE;
     Ans          : CHAR;

BEGIN
  Gotoxy (2,22);
  WRITE (Question, '  ');
  IF Chars = 1 THEN          { Read only one character }
   BEGIN
    Ans:= Readkey;
    Answer := Ans
   END
  ELSE                       { Read until user presses RETURN }
    READLN (Answer);
  FOR Pos := 1 TO LENGTH (Answer) DO
    IF (ORD (Answer[Pos]) >= 97) AND (ORD (Answer[Pos]) <= 122) THEN
      Answer[Pos] := CHR (ORD (Answer[Pos]) - 32);

  Users_Response_To := Answer;
  Gotoxy (2,22);    { Clear the question and answer from the screen }
  Clreol
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
FUNCTION Get_User_Int (Question              : Str80;
                       Min_Int, Max_Int      : INTEGER)  : INTEGER;

VAR  Answer, Err        : INTEGER;
     Answer_Str         : Str12;

BEGIN
  REPEAT
    Answer_Str := Users_Response_To (6, Question);
    VAL (Answer_Str, Answer, Err);
  UNTIL (Err = 0) AND (Answer >= Min_Int) AND (Answer <= Max_Int);
  Get_User_Int := Answer
END;
{---------------------------------------------------------------------}
```

```
{------------------------------------------------------------------------}
FUNCTION Float (INT  : INTEGER)  : REAL;

BEGIN
  Float := INT
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
PROCEDURE Fill_File_Table;

{ This procedure will fill the file table with all the entries  }
{ in the current directory whose system is specified by System_ }
{ Char.                                                          }
   VAR Dirinfo : Searchrec;
          Junk : String;
             D : INTEGER;
BEGIN
  Number_Files_In_Table := 0;
  Findfirst( 'H*.*', Anyfile, Dirinfo);
  WHILE (Doserror = 0) DO
   BEGIN
     IF (Dirinfo.Name[2] IN ['D','N']) THEN BEGIN
          Number_Files_In_Table := Number_Files_In_Table + 1;
          File_Table [Number_Files_In_Table] := Dirinfo.Name;
     END;
     Findnext( Dirinfo);
   END;
   IF Number_Files_In_Table >= File_Table_Size THEN BEGIN
        Number_Files_In_Table:= File_Table_Size;
        Junk := Users_Response_To (1, 'File table overflow - ' +
          'only the first 100 files will be considered (OK)?')
   END;
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
PROCEDURE List_File_Table_To_Screen;

{ List all header files in the table to the screen. }

VAR  File_Number         : INTEGER;

BEGIN
  Clrscr;
  Menu_Head('File Selection');
  Gotoxy (1, 5);
  WRITELN ('Meteor header files on current directory are :');
  WRITELN;
  FOR File_Number := 1 TO Number_Files_In_Table DO
     WRITE ('   [', File_Number :2, '] ', File_Table [File_Number], '    ')
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
PROCEDURE Close_Files;

BEGIN
  CLOSE (Header_File);
  CLOSE (Sample_File)
END;
{------------------------------------------------------------------------}
```

```
{------------------------------------------------------------------------}
PROCEDURE Open_Header_File (     File_Name     : Str14;
                             VAR File_Var      : Header_Fil;
                             VAR File_Present  : BOOLEAN);

VAR  Junk      : String;

BEGIN
  ASSIGN (File_Var, File_Name);
  {$I-}
  RESET (File_Var);
  {$I+}
  File_Present := Ioresult = 0;
  IF NOT File_Present THEN
     Junk := Users_Response_To (1, 'File ' + File_Name +
                ' not on drive (OK)?');
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
PROCEDURE Open_Sample_File (     File_Name     : Str14;
                             VAR File_Var      : Sample_Fil;
                             VAR File_Present  : BOOLEAN);

VAR  Junk       : String;

BEGIN
  ASSIGN (File_Var, File_Name);
  {$I-}
  RESET (File_Var);
  {$I+}
  File_Present := Ioresult = 0;
  IF NOT File_Present THEN
     Junk := Users_Response_To (1, 'File ' + File_Name +
                ' not on drive (OK)?');
END;
{------------------------------------------------------------------------}


{------------------------------------------------------------------------}
PROCEDURE Ready_Files (     Pos_In_Table     : INTEGER;
                        VAR Files_Availible  : BOOLEAN);

VAR  File_Present        : BOOLEAN;
     Sample_Name         : Str12;


BEGIN
  Open_Header_File (File_Table [Pos_In_Table], Header_File, File_Present);
  IF File_Present THEN
   BEGIN
     Sample_Name := File_Table [Pos_In_Table];
     Sample_Name [1] := 'S';
     Open_Sample_File (Sample_Name, Sample_File, File_Present);
     IF File_Present THEN  Files_Availible := TRUE
     ELSE                  Files_Availible := FALSE
   END
  ELSE  Files_Availible := FALSE;
END;
{------------------------------------------------------------------------}
```

```
{--------------------------------------------------------------------}
PROCEDURE Get_Next_Trail;

CONST Bad_Data = 1;

VAR  Sample_Gap, Sample             : INTEGER;
     Real_Gap                       : REAL;
     Sample_Value                   : BYTE;
     Longval : LONGINT;

BEGIN
  READ (Header_File, Hrec);
  Real_Gap := (Hrec.Sample_Range [1] - Hrec.Sample_Range [0]) + 1;
  IF ((Real_Gap > 10000.0) OR (Real_Gap < 1.0))
     OR (Hrec.Noise_Level > 60) OR (Hrec.Noise_Level < 0)
     OR (Hrec.Trail_Type = Bad_Data) THEN BEGIN
     {STUFFED_DATA!! - set trailarray[0] to 0 to avoid crash.}
     Trail_Array[0]:= 0;
     Gotoxy(2,22);
     WRITE('Trail ',Hrec.Trail_Num,' was ill.');
     WRITE(' - ',Hrec.Num_Samples,' Samples,  Noise = ',Hrec.Noise_Level-140)
  END
  ELSE BEGIN
    IF (Hrec.Trail_Type IN Types_Used) THEN BEGIN
       Num_Trails_Considered:= Num_Trails_Considered + 1.0;
       Sample_Gap:= TRUNC (Real_Gap);
       Longval:= TRUNC(Hrec.Sample_Range[0]);
       SEEK (Sample_File, Longval);
       FOR Sample := 1 TO Sample_Gap DO BEGIN
          READ (Sample_File, Sample_Value);
          Trail_Array [Sample] := Sample_Value - 140
      END;
      Trail_Array [0] := Sample_Gap
    END;
 END
END;
{--------------------------------------------------------------------}
```

```
{--------------------------------------------------------------------}
FUNCTION Real_Round (Reel   : REAL)   : REAL;

BEGIN
  Reel := Reel + 0.5;
  Real_Round := Reel - Frac (Reel)
END;
{--------------------------------------------------------------------}
```

```
{--------------------------------------------------------------------}
PROCEDURE Compute_Info_For_Files (Start_File, Stop_File : INTEGER);
  VAR Zero_Before, Files_Availible : BOOLEAN;
          File_Number : INTEGER;
             Numswrit : BYTE;
               Intfil : TEXT;

{--------------------------------------------------------------------}
PROCEDURE Writeit( Num : INTEGER);

BEGIN
   WRITE( Intfil, Num, ' ');
   Numswrit:= Numswrit + 1;
   IF (Numswrit = 25) THEN BEGIN
       Numswrit:= 0;
       WRITELN( Intfil);
   END;
END;
{--------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Update_Info;
  VAR Sig_Noise_Dif, Trailpos : INTEGER;
          CONST   Trail_Mark = -111;

BEGIN
   Chdir( Olddir);
   Writeit( Trail_Mark);     {Marker that new trail beginning.}
   Writeit( Hrec.Trail_Type);
   FOR Trailpos:= 1 TO Trail_Array[0] DO BEGIN
      Sig_Noise_Dif:= Abs( Trail_Array[ Trailpos] - (Hrec.Noise_Level - 140));
      IF (Sig_Noise_Dif < 4)  THEN BEGIN
          IF (NOT Zero_Before) THEN BEGIN
              Zero_Before:= TRUE;
              Writeit( 0)
          END
          ELSE {Do nothing.}
      END
      ELSE BEGIN
          Writeit( Sig_Noise_Dif);
          Zero_Before:= FALSE;
      END;
   END;
   Writeit( 0);
   Zero_Before:= FALSE;
   Chdir( Newpath);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


BEGIN
  Menu_Head('Processing...');
  Chdir( Olddir);
  ASSIGN( Intfil, 'INTFIL');
  REWRITE( Intfil);
  Chdir( Newpath);
  Zero_Before:= FALSE;
  Numswrit:= 0;
  FOR File_Number := Start_File TO Stop_File DO BEGIN
      Ready_Files (File_Number, Files_Availible);
      IF Files_Availible THEN BEGIN
         Gotoxy (5, 7);
         WRITE ('Computing bit rate for file ', File_Table [File_Number]);
         Clreol;
         Gotoxy (5, 9);
         WRITE ('Processing trail ');
         Clreol;
         WHILE NOT (EOF (Header_File)) DO BEGIN
            REPEAT
                Get_Next_Trail;
            UNTIL (((Trail_Array[0] <> 0) AND (Hrec.Trail_Type IN Types_Used))
                   OR (EOF( Header_File)));
            IF (Trail_Array[0] <> 0) AND (Hrec.Trail_Type IN Types_Used) THEN
            BEGIN
               Gotoxy (22, 9);
               WRITE (Hrec.Trail_Num);
               Update_Info;
            END
         END;
         CLOSE( Header_File);
         CLOSE( Sample_File);
      END
      ELSE WRITELN('No files found.');
  END;
  Chdir( Olddir);
  CLOSE( Intfil);
  Chdir( Newpath);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Grab_Types_From_User( VAR Tempset : Typeset);
   VAR Num_In : INTEGER;

BEGIN
   Tempset:= [];
   REPEAT
      Clear_From(22);
      WRITE('Current type selections : ');
      IF (Tempset = []) THEN
           WRITE('Nil')
      ELSE
         FOR Num_In:= 1 TO 30 DO
             IF (Num_In IN Tempset) THEN
                 WRITE(Num_In, ', ');
      WRITE('Please enter next type wanted (1 -> 30, 0 to quit) : ');
      Num_In:= Get_Good_Int( 0, 30);
      IF (Num_In <> 0) THEN
          Tempset:= Tempset + [Num_In];
    UNTIL (Num_In = 0);
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Get_Types( VAR Tempset : Typeset);
  VAR Wish : CHAR;

BEGIN
   Clrscr;
   WRITE('Want All types or a Specified type? (A/S) : ');
   Wish:= Get_Valid_Choice_From(['A','S']);
   IF (Wish = 'A') THEN
       Tempset:= [1..30]
   ELSE BEGIN
       Clrscr;
       Menu_Head('Type Selection');
       WRITELN;
       WRITELN('L = Large type (O/dense + Square roots, Gothics, etc)');
       WRITELN('M = Manual selection of type(s) by numbers');
       WRITELN('O = Overdense type');
       WRITELN('U = Underdense type');
       WRITELN;
       WRITE('Please enter choice : ');
       Wish:= Get_Valid_Choice_From(['L','M','O','U']);
       CASE Wish OF
            'L' : Tempset:= [19, 20, 21, 22, 26, 27, 28];
            'M' : Grab_Types_From_User( Tempset);
            'O' : Tempset:= [20, 21, 26, 27, 28];
            'U' : Tempset:= [5, 9, 10, 11, 12, 13, 14, 16, 17, 24];
       END;
   END;
END;
{-------------------------------------------------------------------------}
```

```
{---------------------------------------------------------------------}
PROCEDURE Auto_Process_Files_In_Directory;

VAR  Junk                                                    : String;
     Start_File, Stop_File                                   : INTEGER;
     Number_Bits_Across, Best_Block_Size, Best_Bit_Rate      : REAL;
     Bit_Index, Block_Index                                  : INTEGER;

BEGIN
  Fill_File_Table;
  IF Number_Files_In_Table = 0 THEN
    Junk := Users_Response_To (1, 'No meteor data files found (OK)?')
  ELSE
   BEGIN
     List_File_Table_To_Screen;
     Start_File := Get_User_Int ('Enter start file number :',
        1, Number_Files_In_Table);
     file_used:= start_file;
     WRITELN( file_used);
     akey:= readkey;
     Stop_File := Get_User_Int ('Enter stop file number :',
        1, Number_Files_In_Table);
     Num_Trails_Considered:= 0.0;
     Get_Types( Types_Used);
     Compute_Info_For_Files (Start_File, Stop_File);
   END
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
PROCEDURE Get_Path;


{---------------------------------------------------------------------}
BEGIN
  Path:= 'C:\WORK\METEOR\DATA';
  Getdir( 0, Olddir);
  REPEAT
    WRITE('Please enter path (default is \work\meteor\data) : ');
    READLN( Newpath);
    IF (Newpath = '') THEN
       Newpath:= Path;
    {$I-} Chdir( Newpath); {$I+}
  UNTIL (Ioresult = 0);
  Path:= Newpath;
END;
{---------------------------------------------------------------------}


{---------------------------------------------------------------------}
PROCEDURE Get_Eb;

BEGIN
   Clrscr;
   WRITE('Please enter Eb to N0 ratio needed (Energy per bit to noise) : ');
   READLN( Eb_To_N0)
END;
{---------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Write_Rates;
  VAR Rat : REAL;
        I : WORD;

BEGIN
  FOR I:= 1 TO 400 DO BEGIN
     Rat:= Rate_Array[I];
     IF (Rat <> 0) THEN BEGIN
         WRITE( Outf, (I*5):10, ' ':9);
         WRITE( Outf, (Rat/1000):6:3,' secs          ');
         WRITE( Outf, (Rat * I * 5):8:0);
         {In Kbits and mSecs so 1000s balance}
         WRITELN( Outf, ' ',(((Rat * I * 5)/Bitsum) * 100.0):10:1);
     END;
  END;
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Show_Results;
  VAR Int_Bitsum, I : LONGINT;

BEGIN
  ASSIGN( Outf, 'BITRES.PRT');
  REWRITE( Outf);
  ASSIGN( Ratf,'ARRAYRES.ABL');
  REWRITE( Ratf);
  WRITELN( Outf, 'File used : ',File_Table[ File_Used]);
  WRITELN( Outf, 'Ratio chosen was ',Eb_To_N0:3:1,' : 1');
  WRITE( Outf, 'Total bits transferred in the hour was : ');
  Int_Bitsum:= ROUND( Bitsum);
  WRITELN( Outf, Int_Bitsum);
  WRITELN( Outf);
  WRITELN( Outf, '      BIT RATE INFORMATION');
  WRITELN( Outf, '      ~~~~~~~~~~~~~~~~~~~');
  WRITELN( Outf, 'Rate (Kbits/sec)   Time used          Bits sent          %');
  Write_Rates;
  WRITE( Ratf, Rate_Array);
  WRITE( Ratf, Bits_Array);
  CLOSE( Ratf);
  CLOSE( Outf);
END;
{-------------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Fill_Sig_Noise_Table;

BEGIN
   Sig_Noise_Table[ 1]:= 0;
   Sig_Noise_Table[ 2]:= 0;
   Sig_Noise_Table[ 3]:= 0;
   Sig_Noise_Table[ 4]:= 0;
   Sig_Noise_Table[ 5]:= 0;
   Sig_Noise_Table[ 6]:= 0;
   Sig_Noise_Table[ 7]:= 0;
   Sig_Noise_Table[ 8]:= 0;
   Sig_Noise_Table[ 9]:= 0;
   Sig_Noise_Table[10]:= 0;
   Sig_Noise_Table[11]:= 0;
   Sig_Noise_Table[12]:= 0;
   Sig_Noise_Table[13]:= 1;
   Sig_Noise_Table[14]:= 1;
   Sig_Noise_Table[15]:= 1;
   Sig_Noise_Table[16]:= 1;
   Sig_Noise_Table[17]:= 2;
   Sig_Noise_Table[18]:= 3;
   Sig_Noise_Table[19]:= 3;
   Sig_Noise_Table[20]:= 5;
   Sig_Noise_Table[21]:= 6;
   Sig_Noise_Table[22]:= 8;
   Sig_Noise_Table[23]:= 11;
   Sig_Noise_Table[24]:= 13;
   Sig_Noise_Table[25]:= 17;
   Sig_Noise_Table[26]:= 22;
   Sig_Noise_Table[27]:= 28;
   Sig_Noise_Table[28]:= 36;
   Sig_Noise_Table[29]:= 44;
   Sig_Noise_Table[30]:= 57;
   Sig_Noise_Table[31]:= 71;
   Sig_Noise_Table[32]:= 90;
   Sig_Noise_Table[33]:= 114;
   Sig_Noise_Table[34]:= 144;
   Sig_Noise_Table[35]:= 181;
   Sig_Noise_Table[36]:= 228;
   Sig_Noise_Table[37]:= 287;
   Sig_Noise_Table[38]:= 362;
   Sig_Noise_Table[39]:= 456;
   Sig_Noise_Table[40]:= 575;
   Sig_Noise_Table[41]:= 723;
   Sig_Noise_Table[42]:= 911;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



BEGIN                    {Main Program}
  Get_Path;
  Get_Eb;
  Bitsum:= 0.0;
  Fillchar( Rate_Array, Sizeof( Rate_Array), 0);
  Fillchar( Bits_Array, Sizeof( Rate_Array), 0);
  Fill_Sig_Noise_Table;
  Auto_Process_Files_In_Directory;
  Chdir( Olddir);
END.
```

Appendix M - Source Listing of the WaitCalc Program

(Gets Wait Time Statistics)

```
{$N+}
PROGRAM Wait_Calculation;
{Takes the strings output from NewWait progam & determines how many}
{times messages could be sent at the different rates.              }
{                    By Stuart Melville.                           }

USES Dos, Crt;


CONST Dbdifs = 42;
      Blocks = 3;

       Bytes10 = 1;
       Bytes80 = 2;
      Bytes256 = 3;

    Trail_Mark = -111;


  VAR Sig_Noise_Gap_Needed : ARRAY[ 1..33] OF INTEGER;
           Samples_Needed : ARRAY[ 1..33, 1..3] OF INTEGER;
              Types_Array : ARRAY[ 0..40, 1..33, 1..3] OF LONGINT;
              Trail_Array : ARRAY[ 1..10000] OF BYTE;
              Pos_Bitrate : ARRAY[ 1..33] OF LONGINT;
              Block_Bytes : ARRAY[ 1..3] OF INTEGER;

         Rate, Block_Spec : INTEGER;
           Blocksize_Bits : LONGINT;
           Bitrate, Msecs : REAL;
            Eo_Trail_File : BOOLEAN;
                Inn, Out : TEXT;
                Cur_Type : INTEGER;


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Fill_Tables;
{NOTE : SIG_NOISE_GAP_NEEDED gives the signal - noise gap required to}
{support the bitrates stored in the corresponding POS_BITRATE tables.}
   VAR I : INTEGER;

BEGIN
   Fillchar( Types_Array, Sizeof( Types_Array), 0);
   Block_Bytes[ 1]:= 10;
   Block_Bytes[ 2]:= 80;
   Block_Bytes[ 3]:= 256;
   FOR I:= 1 TO 4 DO
       Pos_Bitrate[I]:= I;
   Pos_Bitrate[ 5]:= 6;
   Pos_Bitrate[ 6]:= 7;
   Pos_Bitrate[ 7]:= 9;
   Pos_Bitrate[ 8]:= 11;
   Pos_Bitrate[ 9]:= 14;
   Pos_Bitrate[10]:= 18;
   Pos_Bitrate[11]:= 22;
   Pos_Bitrate[12]:= 28;
   Pos_Bitrate[13]:= 35;
   Pos_Bitrate[14]:= 44;
   Pos_Bitrate[15]:= 56;
   Pos_Bitrate[16]:= 70;
   Pos_Bitrate[17]:= 88;
   Pos_Bitrate[18]:= 111;
   Pos_Bitrate[19]:= 140;
   Pos_Bitrate[20]:= 177;
   Pos_Bitrate[21]:= 222;
   Pos_Bitrate[22]:= 280;
   Pos_Bitrate[23]:= 352;
   Pos_Bitrate[24]:= 443;
   Pos_Bitrate[25]:= 558;
   Pos_Bitrate[26]:= 703;
   Pos_Bitrate[27]:= 885;
   Pos_Bitrate[28]:= 1114;
   Pos_Bitrate[29]:= 1402;
   Pos_Bitrate[30]:= 1765;
   Pos_Bitrate[31]:= 2222;
   Pos_Bitrate[32]:= 2798;
```

```
  Pos_Bitrate[33]:= 3522;
  Sig_Noise_Gap_Needed[1]:= 4;
  Sig_Noise_Gap_Needed[2]:= 9;
  Sig_Noise_Gap_Needed[3]:= 11;
  Sig_Noise_Gap_Needed[4]:= 12;
  FOR I:= 5 TO 33 DO      {SIG_NOISE_GAP_NEEDED[5] = S-N for 6Kbps  = 14.}
      Sig_Noise_Gap_Needed[I]:= I + 9;
  Samples_Needed[ 1,1]:= 16;
  Samples_Needed[ 2,1]:= 8;
  Samples_Needed[ 3,1]:= 6;
  Samples_Needed[ 4,1]:= 4;
  Samples_Needed[ 5,1]:= 3;
  Samples_Needed[ 6,1]:= 3;
  Samples_Needed[ 7,1]:= 2;
  Samples_Needed[ 8,1]:= 2;
  Samples_Needed[ 9,1]:= 2;
  FOR I:= 10 TO 33 DO
      Samples_Needed[ I, 1]:= 1;
  Samples_Needed[ 1,2]:= 128;
  Samples_Needed[ 2,2]:= 64;
  Samples_Needed[ 3,2]:= 43;
  Samples_Needed[ 4,2]:= 32;
  Samples_Needed[ 5,2]:= 22;
  Samples_Needed[ 6,2]:= 19;
  Samples_Needed[ 7,2]:= 15;
  Samples_Needed[ 8,2]:= 12;
  Samples_Needed[ 9,2]:= 10;
  Samples_Needed[ 10,2]:= 8;
  Samples_Needed[ 11,2]:= 6;
  Samples_Needed[ 12,2]:= 5;
  Samples_Needed[ 13,2]:= 4;
  Samples_Needed[ 14,2]:= 3;
  Samples_Needed[ 15,2]:= 2;
  Samples_Needed[ 16,2]:= 2;
  Samples_Needed[ 17,2]:= 2;
  FOR I:= 18 TO 33 DO
      Samples_Needed[ I, 2]:= 1;
  Samples_Needed[ 1,3]:= 410;
  Samples_Needed[ 2,3]:= 205;
  Samples_Needed[ 3,3]:= 137;
  Samples_Needed[ 4,3]:= 103;
  Samples_Needed[ 5,3]:= 69;
  Samples_Needed[ 6,3]:= 59;
  Samples_Needed[ 7,3]:= 46;
  Samples_Needed[ 8,3]:= 38;
  Samples_Needed[ 9,3]:= 30;
  Samples_Needed[ 10,3]:= 23;
  Samples_Needed[ 11,3]:= 19;
  Samples_Needed[ 12,3]:= 15;
  Samples_Needed[ 13,3]:= 12;
  Samples_Needed[ 14,3]:= 10;
  Samples_Needed[ 15,3]:= 8;
  Samples_Needed[ 16,3]:= 6;
  Samples_Needed[ 17,3]:= 5;
  Samples_Needed[ 18,3]:= 4;
  Samples_Needed[ 19,3]:= 3;
  Samples_Needed[ 20,3]:= 3;
  Samples_Needed[ 21,3]:= 2;
  Samples_Needed[ 22,3]:= 2;
  Samples_Needed[ 23,3]:= 2;
  FOR I:= 24 TO 33 DO
      Samples_Needed[ I,3]:= 1;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Grab_Trail;
 VAR Pos, Curval : INTEGER;
      Done_Trail : BOOLEAN;

BEGIN
  IF (EOF(Inn)) THEN
     Eo_Trail_File:= TRUE
  ELSE BEGIN
     READ( Inn, Curval);
     IF (Curval = Trail_Mark) THEN BEGIN
          READ( Inn, Curval);
          Cur_Type:= Curval;
          READ( Inn, Curval);
     END;
     Trail_Array[1]:= Curval;
     Pos:= 1;
     Done_Trail:= FALSE;
     WHILE ((NOT (EOF (Inn))) AND (Curval <> 0) AND (NOT Done_Trail)) DO BEGIN
          Pos:= Pos + 1;
          READ( Inn, Curval);
          IF (Curval = Trail_Mark) THEN BEGIN
               READ( Inn, Curval);
               Cur_Type:= Curval;
               Done_Trail:= TRUE;
          END
          ELSE Trail_Array[ Pos]:= Curval;
     END;
     IF (EOF(Inn)) THEN
          Eo_Trail_File:= TRUE;
  END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Find_How_Many_Sends( Whatrate : INTEGER);
   VAR Sig_Needed : INTEGER;
       Num_Sends : ARRAY[1..3] OF LONGINT;
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Check_Succesful_Sends;
  VAR Sent1, Sent2, Sent3 : BOOLEAN;
          Curpos, Numgot : INTEGER;

BEGIN
   Curpos:= 1;
   Sent1:= FALSE;   Sent2:= FALSE;  Sent3:= FALSE;
   Numgot:= 0;
   WHILE ((NOT Sent3) AND (Trail_Array[ Curpos] <> 0)) DO BEGIN
         IF (Trail_Array[ Curpos] < Sig_Needed) THEN
              Numgot:= 0
         ELSE BEGIN
              Numgot:= Numgot + 1;
              IF (Numgot = Samples_Needed[ Whatrate, Bytes10]) THEN
                   Sent1:= TRUE;
              IF (Numgot = Samples_Needed[ Whatrate, Bytes80]) THEN
                   Sent2:= TRUE;
              IF (Numgot = Samples_Needed[ Whatrate, Bytes256]) THEN
                   Sent3:= TRUE;
         END;
         Curpos:= Curpos + 1;
   END;
   IF Sent1 THEN BEGIN
       Num_Sends[ Bytes10]:= Num_Sends[ Bytes10] + 1;
       Types_Array[ Cur_Type, Whatrate, 1]:=
                     Types_Array[ Cur_Type, Whatrate, 1] + 1;
   END;
   IF Sent2 THEN BEGIN
       Num_Sends[ Bytes80]:= Num_Sends[ Bytes80] + 1;
       Types_Array[ Cur_Type, Whatrate, 2]:=
                     Types_Array[ Cur_Type, Whatrate, 2] + 1;
   END;
   IF Sent3 THEN BEGIN
       Num_Sends[ Bytes256]:= Num_Sends[ Bytes256] + 1;
       Types_Array[ Cur_Type, Whatrate, 3]:=
                     Types_Array[ Cur_Type, Whatrate, 3] + 1;
   END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Write_Results;

BEGIN
    WRITE( Out, Pos_Bitrate[ Whatrate]:10, Num_Sends[1]:20);
    WRITELN( Out, Num_Sends[2]:20, Num_Sends[3]:20);
END;


BEGIN
   Sig_Needed:= Sig_Noise_Gap_Needed[ Whatrate];
   Num_Sends[ Bytes10]:= 0;
   Num_Sends[ Bytes80]:= 0;
   Num_Sends[ Bytes256]:= 0;
   REPEAT
       Grab_Trail;
       IF (NOT Eo_Trail_File) THEN
          Check_Succesful_Sends
   UNTIL Eo_Trail_File;
   Write_Results;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Write_Type_Results;
   VAR I, J, K : INTEGER;

BEGIN
   FOR I:= 1 TO 3 DO BEGIN
       WRITELN(Out,'~~~~~~~~ BLOCK ',I,' Results ~~~~~~~~~');
       FOR J:= 1 TO 33 DO BEGIN
           WRITE(Out, 'Rate ',J,': - ');
           FOR K:= 0 TO 40 DO
               IF (Types_Array[ K, J, I] <> 0) THEN
                   WRITE( Out, ' ',K,':', Types_Array[ K, J, I]);
           WRITELN( Out);
       END;
   END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


BEGIN                          {Main Program}
  Fill_Tables;
  ASSIGN( Out, 'RESULTS.WT');
  REWRITE( Out);
  WRITE( Out, 'Bit Rate (Kbps)':19,'10-Byte Block':20);
  WRITELN(Out, '80-Byte Block':20,'256-Byte Block':20);
  FOR Rate:= 1 TO 33 DO BEGIN
       Clrscr;
       WRITELN('Current rate is ',Rate,' out of 33.');
       ASSIGN( Inn, 'INTFIL');
       RESET( Inn);
       Eo_Trail_File:= FALSE;
       Find_How_Many_Sends( Rate);
       CLOSE( Inn);
  END;
  Write_Type_Results;
  CLOSE( Out);
END.
```

Appendix N - Source Listing of the SEER System

```
PROGRAM Network_Simulator;

{This program simulates the operation of a network in the Meteor}
{-Burst Communications environment, allowing the comparison of a}
{number of routing techniques.                                   }
{                                                                }
{By  Stuart Melville.        Version 2.1          February 1991}
{----------------------------------------------------------------}

USES Dos, Crt, Library;

CONST  Numroutes = 5;
         Maxvert = 20;

          Minpth = TRUE;
           Shpth = FALSE;

        Alldelays = 1;
        Lastdelay = 2;

          Maxreal = 99999999999999999999999.0;

        Debugging = FALSE;

           Return = TRUE;
            Noret = FALSE;

            Root3 = 1.73205;

      Trough_Hour = 17;

         Mbc_Mode = FALSE;


TYPE    Delay_Rec = RECORD
                      Mean_Delay, Delay_Deviation : LONGINT;
                                         Basetime : -255..255;
                                           Volume : REAL;
                    END;


          Adjmat = ARRAY[1..Maxvert, 1..Maxvert] OF Delay_Rec;

          Numset = SET OF 1..Maxvert;


        Ppointer = ^pathrecord;

      Pathrecord = RECORD
                     Time_To_Here : REAL;
                           Vertex : INTEGER;
                             Path : Ppointer;
                   END;

       Patharray = ARRAY[1..Maxvert, 1..Maxvert] OF Ppointer;


           Qpointer = ^queuerecord;

        Queuerecord = RECORD
                        Time_To_Here : REAL;
                           Verts_Got : Numset;
                              Vertex : INTEGER;
                                Next : Qpointer;
                                Path : Ppointer;
                      END;

        Route_Res_Array = ARRAY[1..Numroutes] OF REAL;
```

```
          Str80 = String[80];
          Str40 = String[40];
          Str12 = String[12];


VAR   Shortpath, Minpath, Userpath : Patharray;
            Good_Rate, Error_Rate : REAL;
             Num_Verts, Num_Links : LONGINT;
                    Quickest_Done : REAL;
             Sim_Hours, Mean_Flow : LONGINT;
                    Num_Messages : LONGINT;
                    Selected_Set : Numset;
                    Network_Held : BOOLEAN;
                       Delay_Tot : ARRAY[ 1..Numroutes, Alldelays..Lastdelay]
                                                                    OF REAL;
                      Delay_Sqrs  : Route_Res_Array;
                Delay_Mean, Delay_Std : Route_Res_Array;
                    Current_Hour : INTEGER;
                      Base_Delay : LONGINT;
                       Base_Hour : INTEGER;
                        Messages : ARRAY[1..Maxvert, 1..Maxvert] OF WORD;
                         Routing : ARRAY[1..Numroutes] OF Str40;
                         Network : Adjmat;
                          Anykey : CHAR;
                          Volume : REAL;
                            Fnam : Str12;
                             Out : TEXT;


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION radians(x : REAL) : REAL;

BEGIN
  radians := pi * x / 180.0
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Debug( STR : Str80; C_Return : BOOLEAN);

BEGIN
   IF Debugging THEN
      IF C_Return THEN
           WRITELN( STR)
      ELSE WRITE(STR);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Show_Intro_Screen;

BEGIN
   Clrscr;
   WRITELN('System for Evaluation of Efficiency of Routings (SEER)');
   WRITELN('by Stuart Melville.                                Ver 2.1');

   Gotoxy(1,22);
   WRITE('Any key to continue...');
   Anykey:= Readkey;
   Clrscr;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Initialise_Route_Names;

BEGIN
   Routing[1]:= 'Minimum Hop Routing';
   Routing[2]:= 'Shortest Path Routing';
   Routing[3]:= 'Flood Routing';
   Routing[4]:= 'Changing Network Routing';
   Routing[5]:= 'User-defined Routing';
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Initialise;

BEGIN
   Show_Intro_Screen;
   ASSIGN( Out, 'RESULTS.SIM');
   REWRITE( Out);
   Network_Held:= FALSE;
   Initialise_Route_Names;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Get_Correct_File : Str12;

BEGIN
   Clrscr;
   WRITE('Please enter file name for network links and delays : ');
   READLN( Fnam);
   Get_Correct_File:= Fnam;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Get_Hour_Degrees( Hour_In : BYTE) : INTEGER;

BEGIN
   Get_Hour_Degrees:= ((Abs( (Hour_In + 6) - Trough_Hour)) MOD 24) * 15;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Get_Volume( Base_Delay : REAL; Base_Hour : INTEGER) : REAL;

BEGIN
  IF Debugging THEN BEGIN
      WRITELN('Base delay is ',Base_Delay,'  base hour is ',Base_Hour);
      Debug('',TRUE);
  END;
(BaseGet_Hour)Volume:=);Base_Delay / (2.5 + (1.5 * Sin ( radians(Get_Hour_Degrees
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Load_Network_And_Delays;
  VAR V1, V2, I, J, Delay, Deviation : LONGINT;
                           Cur_Rec : Delay_Rec;
                             Btime : INTEGER;
                              Fnam : Str12;
                               Inn : TEXT;

BEGIN
   Fnam:= Get_Correct_File;
   ASSIGN( Inn, Fnam);
   RESET( Inn);
   READLN( Inn, Num_Verts);
   READLN( Inn, Num_Links);
   FOR I:= 1 TO Num_Verts DO
       FOR J:= 1 TO Num_Verts DO BEGIN
           Network[I,J].Mean_Delay:= -1;
           Network[I,J].Delay_Deviation:= -1;
           Network[I,J].Basetime:= -1;
       END;
   FOR I:= 1 TO Num_Links DO BEGIN
       READ( Inn, V1);
       READ( Inn, V2);
       READ( Inn, Delay);
       READ( Inn, Deviation);
       READLN( Inn, Btime);
       Cur_Rec.Mean_Delay:= Delay;
       Cur_Rec.Delay_Deviation:= Deviation;
       Cur_Rec.Basetime:= Btime;
       Cur_Rec.Volume:= Get_Volume( Delay, Btime);
       Network[ V1, V2]:= Cur_Rec;
   END;
   Network_Held:= TRUE;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Show_Available_Routings;
   VAR I : INTEGER;

BEGIN
   Clrscr;
   Inverse;
   Write_At( 20,1, 'Available Routing Algorithms');
   Write_At( 20,2, '----------------------------');
   Normal;
   FOR I:= 1 TO Numroutes DO BEGIN
       Gotoxy( 2, I + 5);
       IF (I IN Selected_Set) THEN
           Inverse;
       WRITE(I,'  =  ',Routing[I]);
       Normal;
   END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Int_From_Char( Charin : CHAR) : INTEGER;

BEGIN
   Int_From_Char:= ORD( Charin) - 48;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{---------------------------------------------------------------------}
PROCEDURE Get_Routings_To_Consider;
  VAR Curkey : CHAR;
       Numin : BYTE;

BEGIN
   Selected_Set:= [];
   REPEAT
      Show_Available_Routings;
      Gotoxy(1,19);
      WRITE('Currently selected strategies shown by highlight');
      Gotoxy(1,20);
      WRITE('Enter next selection - ');
      IF (Selected_Set <> []) THEN
          WRITE('or D if done : ');
      IF (Selected_Set = []) THEN
           Curkey:= Get_Valid_Choice_From(['1'..'5'])
      ELSE Curkey:= Get_Valid_Choice_From(['D','1'..'5']);
      IF (Curkey <> 'D') THEN BEGIN
           Numin:= Int_From_Char( Curkey);
           Selected_Set:= Selected_Set + [Numin];
      END;
   UNTIL ((Selected_Set <> []) AND (Curkey = 'D'));
END;
{---------------------------------------------------------------------}



{---------------------------------------------------------------------}
PROCEDURE Get_Other_Parameters;

BEGIN
   Clrscr;
   Menu_Head('Parameter Entry');
   WRITE('Please enter error rate (0 -> 0.99) : ');
   READLN( Error_Rate);
   Good_Rate:= 1.0 - Error_Rate;
   WRITE('Please enter simulation period in hours (1 -> 1000) : ');
   READLN( Sim_Hours);
   WRITE('Please enter mean number of messages per hour in network : ');
   READLN( Mean_Flow);
END;
{---------------------------------------------------------------------}



{---------------------------------------------------------------------}
PROCEDURE Set_Up_Routes( Fromwhat : BOOLEAN;  VAR Patharr : Patharray);
  VAR Startv, I, Smallest : INTEGER;
              Path_Exists : BOOLEAN;
                  Pathmat : ARRAY[1..Maxvert, 1..Maxvert] OF BYTE;
                    Count : INTEGER;
                     Cost : ARRAY[ 1..Maxvert, 1..Maxvert] OF INTEGER;
                    Found : ARRAY[1..Maxvert] OF BOOLEAN;
                     Dist : ARRAY[1..Maxvert] OF INTEGER;
                      Min : LONGINT;


      PROCEDURE Set_Up_Cost_Matrix( Forwhat : BOOLEAN);
         VAR I, J : INTEGER;

      BEGIN
         FOR I:= 1 TO Num_Verts DO
            FOR J:= 1 TO Num_Verts DO
               Cost[ I,J]:= -1;
         FOR I:= 1 TO Num_Verts DO
            FOR J:= 1 TO Num_Verts DO
               IF (Network[I,J].Mean_Delay <> -1) THEN  {Link exists}
                  IF (Forwhat = Minpth) THEN
                      Cost[I,J]:= 1
                  ELSE Cost[I,J]:= Network[I,J].Mean_Delay;
      END;
```

```
FUNCTION Get_Smallest : BOOLEAN;
   VAR I : INTEGER;

BEGIN
   Min:= Maxlongint;
   FOR I:= 1 TO Num_Verts DO
      IF ((NOT Found[I]) AND (Dist[I] < Min) AND
                             (Dist[I] <> -1)) THEN BEGIN
            Min:= Dist[I];
            Smallest:= I;
      END;
   Get_Smallest:= (Min <> Maxlongint);
END;


PROCEDURE Get_Path_List_From_Array( Startv : INTEGER);
   VAR Curstop, Pathfinder : INTEGER;
            Ppoint, Runner : Ppointer;


      PROCEDURE Add_On_List;

      BEGIN
          NEW( Ppoint);
          Ppoint^.Path:= NIL;
          IF (Pathmat[ Curstop, Pathfinder] <> 0) THEN
               Ppoint^.Vertex:= Pathmat[ Curstop, Pathfinder]
          ELSE Ppoint^.Vertex:= Curstop;
          Runner:= Patharr[ Startv, Curstop];
          IF (Runner = NIL) THEN
              Patharr[ Startv, Curstop]:= Ppoint
          ELSE BEGIN
               WHILE (Runner^.Path <> NIL) DO
                   Runner:= Runner^.Path;
              Runner^.Path:= Ppoint;
          END;
      END;


BEGIN
  FOR Curstop:= 1 TO Num_Verts DO BEGIN
     Patharr[ Startv, Curstop]:= NIL;
     IF (Dist[ Curstop] <> -1) THEN BEGIN
         Pathfinder:= 1;
         IF Debugging THEN
             WRITE('path from ',startv,' to ', curstop,' = ');
         WHILE (Pathmat[ Curstop, Pathfinder] <> 0) DO BEGIN
             Add_On_List;
             IF Debugging THEN
                  WRITE( pathmat[ curstop, pathfinder]);
             Pathfinder:= Pathfinder + 1;
         END;
         IF Debugging THEN BEGIN
             WRITELN;
             Anykey:= Readkey;
         END;
         Add_On_List;
     END;
  END;
END;
```

```
      PROCEDURE Add_To_Path( Vert : INTEGER);
         VAR Curpos : INTEGER;

      BEGIN
         Curpos:= 1;
         WHILE( Pathmat[ Smallest, Curpos] <> 0) DO BEGIN
             Pathmat[ Vert, Curpos]:= Pathmat[ Smallest, Curpos];
             Curpos:= Curpos + 1;
         END;
         Pathmat[ Vert, Curpos]:= Smallest;
         Pathmat[ Vert, (Curpos + 1)]:= 0;
      END;



BEGIN                                  {Of SET_UP_ROUTES procedure}
  IF Debugging THEN
       WRITELN('in set_up_routes');
  FOR Startv:= 1 TO Num_Verts DO BEGIN
     Fillchar( Pathmat, Sizeof( Pathmat), 0);
     Set_Up_Cost_Matrix( Fromwhat);
     FOR I:= 1 TO Num_Verts DO
        Dist[I]:= Cost[ Startv, I];
     Fillchar( Found, Sizeof( Found), FALSE);
     Found[ Startv]:= TRUE;
     FOR Count:= 1 TO (Num_Verts - 1) DO BEGIN
       Path_Exists:= Get_Smallest;
       IF Path_Exists THEN BEGIN
          Found[ Smallest]:= TRUE;
          FOR I:= 1 TO Num_Verts DO
             IF (NOT Found[I]) THEN BEGIN
               IF (Cost[Smallest,I] <> -1) THEN
                    IF (Dist[I] = -1) THEN BEGIN
                       Add_To_Path( I);
                       Dist[I]:= Min + Cost[Smallest,I];
                    END
                    ELSE
                       IF (Dist[I] > (Min + Cost[Smallest,I])) THEN BEGIN
                          Add_To_Path( I);
                          Dist[I]:= Min + Cost[Smallest,I];
                       END;
             END;
       END;
     END;
     Get_Path_List_From_Array( Startv);
  END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-------------------------------------------------------------------}
PROCEDURE Get_User_Routes;
   VAR Ppoint, Runner : Ppointer;
       V1, V2, Vert : INTEGER;


BEGIN
   FOR V1:= 1 TO Num_Verts DO
      FOR V2:= 1 TO Num_Verts DO
         Userpath[ V1, V2]:= NIL;
      IF (V1 <> V2) THEN
         REPEAT
            WRITE('From vertex ',V1,', to ',V2,' enter next in path : ');
            READLN( Vert);
            IF (Vert <> 0) THEN BEGIN
               NEW( Ppoint);
               Ppoint^.Path:= NIL;
               Ppoint^.Vertex:= Vert;
               Runner:= Userpath[ V1, V2];
               IF (Runner = NIL) THEN
                  Userpath[ V1, V2]:= Ppoint
               ELSE BEGIN
                  WHILE (Runner^.Path <> NIL) DO
                     Runner:= Runner^.Path;
                  Runner^.Path:= Ppoint;
               END;
            END;
         UNTIL (Vert = 0);
END;
{-------------------------------------------------------------------}


{-------------------------------------------------------------------}
PROCEDURE Give_Routes;
   VAR Runner : Ppointer;
       V1, V2 : WORD;

   PROCEDURE Give_Path( Nam : Str12; P : Ppointer);
       VAR Vert : BYTE;

   BEGIN
       WRITE( Nam, V1,'->',V2,' :');
       IF (P = NIL) THEN
           WRITELN('No path')
       ELSE BEGIN
          WHILE (P <> NIL) DO BEGIN
             WRITE(P^.Vertex,',');
             P:= P^.Path;
          END;
          WRITELN;
       END;
       Anykey:= Readkey;
   END;

BEGIN
   Clrscr;
   WRITELN('In give routes');
   FOR V1:= 1 TO Num_Verts DO
      FOR V2:= 1 TO Num_Verts DO BEGIN
         IF (1 IN Selected_Set) THEN BEGIN
            Runner:= Minpath[ V1, V2];
            Give_Path('Minpath : ',Runner);
         END;
         IF (2 IN Selected_Set) THEN BEGIN
            Runner:= Shortpath[ V1, V2];
            Give_Path('Shortpath : ',Runner);
         END;
         IF (5 IN Selected_Set) THEN BEGIN
            Runner:= Userpath[ V1, V2];
            Give_Path('Userpath : ',Runner);
         END
      END;
END;
{-------------------------------------------------------------------}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Set_Up_Routing_Tables;

BEGIN
   IF (1 IN Selected_Set) THEN
       Set_Up_Routes( Minpth, Minpath);
   IF (2 IN Selected_Set) THEN
       Set_Up_Routes( Shpth, Shortpath);
   IF (5 IN Selected_Set) THEN
       Get_User_Routes;
   IF Debugging THEN
      Give_Routes;
END;
{-----------------------------------------------------------------------}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Assign_Messages_Sources_And_Destinations;
   VAR I, Source, Dest : INTEGER;


BEGIN
   Fillchar( Messages, Sizeof( Messages), 0);
   Num_Messages:= Mean_Flow * Sim_Hours;
   FOR I:= 1 TO Num_Messages DO BEGIN
       REPEAT
          Source:= Random( Num_Verts) + 1;
          Dest:= Random( Num_Verts) + 1;
       UNTIL ( Source <> Dest);
       Messages[ Source, Dest]:= Messages[ Source, Dest] + 1;
   END;
END;
{-----------------------------------------------------------------------}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Initialise_Route_Delay_Counters;
   VAR Rout : BYTE;

BEGIN
   FOR Rout:= 1 TO Numroutes DO BEGIN
       Delay_Tot[ Rout, Alldelays]:= 0.0;
       Delay_Sqrs[ Rout]:= 0.0;
   END;
END;
{-----------------------------------------------------------------------}
```

```
{-----------------------------------------------------------------------}
FUNCTION Get_Current_Mean( V1, V2, Hour_In : BYTE) : REAL;
    VAR Hour_Degrees : INTEGER;

BEGIN
   Hour_Degrees:= Get_Hour_Degrees( Hour_In);
   Get_Current_Mean:= Network[V1,V2].Volume * (2.5 +
                              (1.5 * Sin( radians(Hour_Degrees))));
END;
{-----------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Expected_Time( V1, V2 : INTEGER) : REAL;
  VAR Expected_Delay, Deviation_Added : REAL;

BEGIN
   IF Debugging THEN BEGIN
       Gotoxy( 1, 10);
       WRITELN('Sending message from ',v1,' to ', v2);
   END;
   Expected_Delay:= Get_Current_Mean( V1, V2, Current_Hour);
   Deviation_Added:= Random * (Network[ V1,V2].Delay_Deviation * Root3);
   IF (Random > 0.5) THEN BEGIN
        IF (Deviation_Added > Expected_Delay) THEN
             Expected_Delay:= 0.0
        ELSE Expected_Delay:= Expected_Delay - Deviation_Added
   END
   ELSE Expected_Delay:= Expected_Delay + Deviation_Added;
   Expected_Time:= Expected_Delay;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Message_Fails : BOOLEAN;

BEGIN
   Message_Fails:= (Random > Good_Rate);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Time_Over_Link( V1, V2 : INTEGER) : REAL;
   VAR Total_Delay : REAL;

BEGIN
   Total_Delay:= Expected_Time( V1,V2);
   WHILE Message_Fails DO
       Total_Delay:= Total_Delay + Expected_Time(V1,V2);
   Time_Over_Link:= Total_Delay;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Time_Over_Path( Start : INTEGER; Pth : Ppointer) : REAL;
   VAR Current, Nextup : INTEGER;
       Total_Path_Time : REAL;
           End_Reached : BOOLEAN;
                Runner : Ppointer;

BEGIN
   Current:= Start;
   Nextup:= Pth^.Vertex;
   Runner:= Pth;
   Total_Path_Time:= 0.0;
   End_Reached:= FALSE;
   REPEAT
      Total_Path_Time:= Total_Path_Time + Time_Over_Link( Current, Nextup);
      IF (Runner^.Path <> NIL) THEN BEGIN
          Current:= Nextup;
          Runner:= Runner^.Path;
          Nextup:= Runner^.Vertex;
      END
      ELSE End_Reached:= TRUE;
   UNTIL End_Reached;
   Time_Over_Path:= Total_Path_Time;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Send_On_Some_Route( V1, V2 : INTEGER; Routing : BYTE);
  VAR There_Back_Time : REAL;
           Pathwanted : Ppointer;

BEGIN
   CASE Routing OF
        1 : Pathwanted:= Minpath[ V1,V2];
        2 : Pathwanted:= Shortpath[ V1,V2];
        5 : Pathwanted:= Userpath[ V1,V2];
   END;
   There_Back_Time:= Time_Over_Path( V1, Pathwanted);          {Message sent}

   CASE Routing OF
        1 : Pathwanted:= Minpath[ V2,V1];
        2 : Pathwanted:= Shortpath[ V2,V1];
        5 : Pathwanted:= Userpath[ V2,V1];
   END;
   There_Back_Time:= There_Back_Time + Time_Over_Path( V2, Pathwanted);
                                                                 {Ack sent}
   Delay_Tot[ Routing, Lastdelay]:= There_Back_Time;
   Delay_Tot[ Routing, Alldelays]:=
                        Delay_Tot[ Routing, Alldelays] +  There_Back_Time;
   Delay_Sqrs[ Routing]:= Delay_Sqrs[ Routing] + Sqr( There_Back_Time);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Set_Up_Root( V1 : INTEGER; VAR Queue : Qpointer);

BEGIN
   NEW( Queue);
   Queue^.Next:= NIL;
   Queue^.Path:= NIL;
   Queue^.Vertex:= V1;
   Queue^.Time_To_Here:= 0.0;
   Queue^.Verts_Got:= [V1];
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Get_Front_Path( VAR Front, Prior : Qpointer; Queue : Qpointer);

BEGIN
  Front:= Queue;
  Prior:= Queue;
  WHILE (Front^.Next <> NIL) DO BEGIN
     Prior:= Front;
     Front:= Front^.Next;
  END;
  debug(' front path got', return);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-----------------------------------------------------------------------}
PROCEDURE Get_Vertices_On_Current_Path( Front : Qpointer; VAR Pset : Numset);
   VAR Runner : Ppointer;

BEGIN
   Pset:= [];
   Pset:= Pset + [Front^.Vertex];
   Runner:= Front^.Path;
   IF Debugging THEN
      WRITE('Vertices on current path are ',Front^.Vertex);
   WHILE (Runner <> NIL) DO BEGIN
      Pset:= Pset + [Runner^.Vertex];
      IF Debugging THEN
         WRITE(' ',Runner^.vertex);
      Runner:= Runner^.Path
   END;
   IF Debugging THEN BEGIN
      WRITELN;
      anykey:= readkey;
   END
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
FUNCTION Link_Exists( Head : Qpointer; Nextv : INTEGER) : BOOLEAN;
   VAR Runner : Ppointer;
       Oldv : WORD;

BEGIN
   IF (Head^.Path = NIL) THEN
      Oldv:= Head^.Vertex
   ELSE BEGIN
      Runner:= Head^.Path;
      WHILE (Runner^.Path <> NIL) DO
           Runner:= Runner^.Path;
      Oldv:= Runner^.Vertex;
   END;
   Link_Exists:= (Network[ Oldv, Nextv].Mean_Delay <> -1);
END;
{-----------------------------------------------------------------------}


{-----------------------------------------------------------------------}
FUNCTION Get_Path_Time( Head : Qpointer; Nextv : INTEGER) : REAL;
   VAR Pathp : Ppointer;
       Oldv : WORD;

BEGIN
     IF (Head^.Path = NIL) THEN
        Oldv:= Head^.Vertex
     ELSE BEGIN
        Pathp:= Head^.Path;
        WHILE (Pathp^.Path <> NIL) DO
             Pathp:= Pathp^.Path;
       Oldv:= Pathp^.Vertex;
     END;
     Get_Path_Time:= Head^.Time_To_Here + Time_Over_Link( Oldv, Nextv);
END;
{-----------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Make_Duplicate( VAR Front, Newpath : Qpointer;
                                                    VAR Lastone : Ppointer);

   VAR Newnode, Newrun, Runner : Ppointer;

BEGIN
   NEW( Newpath);
   IF Debugging THEN
      WRITE('New path from ',Front^.vertex, ' : ');
   Newpath^.Vertex:= Front^.Vertex;
   Newpath^.Verts_Got:= Front^.Verts_Got;
   IF (Front^.Path = NIL) THEN BEGIN
      Newpath^.Path:= NIL;
      Lastone:= NIL;
   END
   ELSE BEGIN
      Runner:= Front^.Path;
      NEW( Newnode);
      Newnode^.Vertex:= Runner^.Vertex;
      IF Debugging THEN
         WRITE(newnode^.vertex,' ');
      Newpath^.Path:= Newnode;
      Newrun:= Newnode;
      IF (Runner^.Path = NIL) THEN BEGIN
          Newrun^.Path:= NIL;
          Lastone:= Newrun;
      END
      ELSE BEGIN
         REPEAT
            Runner:= Runner^.Path;
            NEW( Newnode);
            Newnode^.Vertex:= Runner^.Vertex;
            IF Debugging THEN
                WRITE( newnode^.vertex, ' ');
            Newrun^.Path:= Newnode;
            Newrun:= Newnode;
         UNTIL (Runner^.Path = NIL);
         Newnode^.Path:= NIL;
         Lastone:= Newnode;
      END;
   END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}




{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Set_New_Path( VAR Front, Prior : Qpointer; Newv : INTEGER;
                                                       Tottime : REAL);
   VAR Lastnode, Added_Node : Ppointer;
                    Newpath : Qpointer;

BEGIN
   Make_Duplicate( Front, Newpath, Lastnode);
   Newpath^.Verts_Got:= Newpath^.Verts_Got + [Newv];
   NEW( Added_Node);
   Added_Node^.Vertex:= Newv;
   Added_Node^.Time_To_Here:= Tottime;
   IF Debugging THEN
      WRITELN( added_node^.vertex);
   Added_Node^.Path:= NIL;
   IF (Lastnode  = NIL) THEN
        Newpath^.Path:= Added_Node
   ELSE Lastnode^.Path:= Added_Node;
   Newpath^.Time_To_Here:= Tottime;
   Prior^.Next:= Newpath;   {Add to queue of paths - FRONT pointer to be}
                            {deleted once all neighbours added, so there}
                            {is no worry about 'losing' it.             }
   Newpath^.Next:= NIL;
   Prior:= Newpath;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Check_Quicker_To_I ( I : WORD; VAR Tott : REAL; Queue : Qpointer);
   VAR Pathlook : Ppointer;
         Runner : Qpointer;

BEGIN
   Runner:= Queue;
   REPEAT
      IF (I IN Runner^.Verts_Got) THEN BEGIN
           Pathlook:= Runner^.Path;
           WHILE (Pathlook^.Vertex <> I) DO
                Pathlook:= Pathlook^.Path;
           IF (Pathlook^.Time_To_Here <= Tott) THEN  {Duplicate arriving  }
                Tott:= Maxreal;                      {Ensures path ignored}
      END;
      Runner:= Runner^.Next;
   UNTIL ((Tott = Maxreal) OR (Runner = NIL));
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Add_Neighbours( VAR Front, Prior, Queue : Qpointer; V2 : INTEGER;
                                                VAR Neighbour_Added : BOOLEAN);
   VAR Pathset : Numset;
      Tot_Time : REAL;
             I : WORD;

BEGIN
   Neighbour_Added:= FALSE;
   Pathset:= Front^.Verts_Got;
   FOR I:= 1 TO Num_Verts DO
       IF (Link_Exists( Front, I) AND (NOT (I IN Pathset))) THEN BEGIN
            Tot_Time:= Get_Path_Time( Front, I);
            Check_Quicker_To_I( I, Tot_Time, Queue);
            IF (Tot_Time < Quickest_Done) THEN
                 IF (I = V2) THEN
                      Quickest_Done:= Tot_Time
                 ELSE BEGIN
                      Set_New_Path( Front, Prior, I, Tot_Time);
                      Neighbour_Added:= TRUE;
                 END;
       END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}



{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Wipeout( VAR Front, Queue : Qpointer);
   VAR Marker, Backstep : Ppointer;
                Que_Run : Qpointer;

BEGIN
   IF (Queue = Front) THEN
        Queue:= Front^.Next
   ELSE BEGIN
        Que_Run:= Queue;
        WHILE ((Que_Run^.Next <> Front) AND (Que_Run^.Next <> NIL)) DO
              Que_Run:= Que_Run^.Next;
        Que_Run^.Next:= NIL;
   END;
   Marker:= Front^.Path;
   DISPOSE( Front);
   WHILE (Marker <> NIL) DO BEGIN
      Backstep:= Marker;
      Marker:= Marker^.Path;
      DISPOSE( Backstep);
   END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Sort_Paths_By_Length( VAR Queue : Qpointer);
   VAR Sorted, Noswops : BOOLEAN;
         Prior, N1, N2 : Qpointer;


    PROCEDURE Swop( Prior, Node1, Node2 : Qpointer);
       VAR Temp : Qpointer;

    BEGIN
       IF (Node1 = Queue) THEN BEGIN
           Queue:= Node2;
           Temp:= Node2^.Next;
           Node2^.Next:= Node1;
           Node1^.Next:= Temp;
       END
       ELSE BEGIN
           Prior^.Next:= Node2;
           Temp:= Node2^.Next;
           Node2^.Next:= Node1;
           Node1^.Next:= Temp;
       END;
    END;

BEGIN                                   {Of Sort_Paths_By_Length routine}
   Sorted:= FALSE;
   IF (Queue = NIL) THEN
       Sorted:= TRUE
   ELSE
       IF (Queue^.Next = NIL) THEN      {Single-element list}
           Sorted:= TRUE;
   WHILE (NOT Sorted) DO BEGIN
       Noswops:= TRUE;
       Prior:= Queue;
       N1:= Queue;
       N2:= Queue^.Next;
       REPEAT
          IF (N1^.Time_To_Here < N2^.Time_To_Here) THEN BEGIN
              Swop( Prior, N1, N2);
              Noswops:= FALSE;
              Prior:= N2;
              N2:= N1^.Next;  {Swopped N1 stays as N1 for next comparison}
          END
          ELSE BEGIN
              Prior:= N1;
              N1:= N2;
              N2:= N2^.Next;
          END;
       UNTIL (N2 = NIL);
       IF Noswops THEN
          Sorted:= TRUE;
   END;
END;
{-------------------------------------------------------------------------}



{-------------------------------------------------------------------------}
FUNCTION Not_Empty( Queue : Qpointer) : BOOLEAN;

BEGIN
   Not_Empty:= (Queue <> NIL);
END;
{-------------------------------------------------------------------------}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION Branch_And_Bound( V1, V2 : INTEGER) : REAL;
   VAR Queue, Front, Prior : Qpointer;
           Neighbour_Added : BOOLEAN;

BEGIN
   Quickest_Done:= Maxreal;
   debug('b and b : ', noret);
   IF Debugging THEN
       WRITELN(' from ',v1, ' to ', v2);
   Set_Up_Root( V1, Queue);
   WHILE Not_Empty( Queue) DO BEGIN
       Get_Front_Path( Front, Prior, Queue);
       Add_Neighbours( Front, Prior, Queue, V2, Neighbour_Added);
       Wipeout( Front, Queue);
       Sort_Paths_By_Length( Queue);
   END;
   IF Debugging THEN BEGIN
       WRITE('quickest was ',quickest_done:1:2);
       anykey:= readkey;
   END;
   Branch_And_Bound:= Quickest_Done;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Send_On_Flood( V1, V2 : INTEGER);
  VAR Quickest_Time : REAL;
         Other_Rout : BYTE;

BEGIN
   debug( 'into flood', Return);
   Quickest_Time:= Branch_And_Bound( V1, V2) + Branch_And_Bound( V2, V1);
                   {Message + Ack.}
   FOR Other_Rout:= 1 TO Numroutes DO
      IF ((Other_Rout IN Selected_Set) AND (Other_Rout <> 3)) THEN
          IF (Delay_Tot[ Other_Rout, Lastdelay] < Quickest_Time) THEN
              Quickest_Time:= Delay_Tot[ Other_Rout, Lastdelay];
   Delay_Tot[ 3, Lastdelay]:= Quickest_Time;
   Delay_Tot[ 3, Alldelays]:= Delay_Tot[ 3, Alldelays]  + Quickest_Time;
   Delay_Sqrs[ 3]:= Delay_Sqrs[ 3] + Sqr( Quickest_Time);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Send_A_Message( V1, V2 : INTEGER);
  VAR I : BYTE;

BEGIN
  FOR I:= 1 TO Numroutes DO
      IF (I IN Selected_Set) THEN
           IF (I = 3) THEN
                Send_On_Flood( V1, V2)
           ELSE Send_On_Some_Route( V1, V2, I);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Compute_Totals;
  VAR Delay_Var : Route_Res_Array;
              I : INTEGER;

BEGIN
  FOR I:= 1 TO Numroutes DO
      IF (I IN Selected_Set) THEN BEGIN
          Delay_Mean[I]:= (Delay_Tot[ I, Alldelays]/Num_Messages);
          Delay_Var[I]:= (Delay_Sqrs[I] -
                            ( Sqr ( Delay_Tot[ I,Alldelays] ) / Num_Messages))
                           / (Num_Messages - 1);
          Delay_Std[I]:= Sqrt( Delay_Var[I]);
      END;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Update_Percent_Complete_Info( VAR Num_Sent : LONGINT);
   VAR Pc_Sent : WORD;

BEGIN
   Num_Sent:= Num_Sent + 1;
   Pc_Sent:= TRUNC( (Num_Sent/Num_Messages) * 100.0);
   Gotoxy( 20, 11);
   WRITE( Pc_Sent:3);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Simulate_Network_Operations;
   VAR Sorce, Dest, Num_Mess, Messages_Sent : LONGINT;

BEGIN
   Assign_Messages_Sources_And_Destinations;
   Initialise_Route_Delay_Counters;
   Messages_Sent:= 0;
   Gotoxy( 22, 11);
   WRITE('0% Completed');
   FOR Sorce:= 1 TO Num_Verts DO
       FOR Dest:= 1 TO Num_Verts DO
           IF (Messages[ Sorce, Dest] <> 0) THEN BEGIN
               Base_Delay:= Network[ Sorce, Dest].Mean_Delay;
               FOR Num_Mess:= 1 TO Messages[ Sorce, Dest] DO BEGIN
                  IF Mbc_Mode THEN
                       Current_Hour:= Random( 24) + 1
                  ELSE Current_Hour:= Base_Hour;
                  Send_A_Message( Sorce, Dest);
                  Update_Percent_Complete_Info( Messages_Sent);
               END;
           END;
   Compute_Totals;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Write_Results;
  VAR I, J : BYTE;

BEGIN
   WRITELN(Out,'_________________________________________________');
   WRITELN( Out);
   WRITELN( Out,'Results of Routing Simulation');
   WRITELN( Out,'-----------------------------');
   WRITELN( Out);
   WRITELN( Out,'Using data from file ',Fnam);
   WRITELN( Out,' Parameters : ');
   WRITELN( Out,'              Error Rate        = ',Error_Rate:1:3);
   WRITELN( Out,'              Simulation period = ',Sim_Hours,' hours');
   WRITE( Out,'              Traffic flow      = ');
   wRITELN( Out, Mean_Flow,' messages per hcur');
   WRITELN( Out);
   WRITELN( Out);
   WRITELN( Out,'            RESULTS');
   WRITELN( Out,'            =======');
   WRITELN( Out);
   WRITELN( Out,'                                 Mean Delay      Delay Deviation');
   FOR I:= 1 TO Numroutes DO
       IF (I IN Selected_Set) THEN BEGIN
            WRITE( Out, Routing[I]);
            FOR J:= LENGTH(Routing[I]) TO 30 DO
                 WRITE( Out,' ');
            WRITELN( Out, Delay_Mean[I]:12:2,Delay_Std[I]:19:2);
       END;
   WRITELN(Out);
   WRITELN(Out,'_________________________________________________');
   WRITELN( Out);
   WRITELN( Out);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
FUNCTION No_More_Processing_Wanted : BOOLEAN;

BEGIN
   Clrscr;
   Gotoxy( 12, 13);
   WRITE('Want another run/another network (Y/N) ? ');
   No_More_Processing_Wanted:= (Get_Valid_Choice_From(['Y','N']) = 'N');
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Process;

BEGIN
   REPEAT
      Load_Network_And_Delays;
      Get_Routings_To_Consider;
      Get_Other_Parameters;
      Set_Up_Routing_Tables;
      Simulate_Network_Operations;
      Write_Results;
   UNTIL No_More_Processing_Wanted;
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Finalise;

BEGIN
   CLOSE( Out);
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


BEGIN                    {Main Program}
  Initialise;
  Process;
  Finalise;
END.
```

Appendix O - Source Listing of the Gen_Conn Program

(Generates networks of different connectivities)

```
PROGRAM Gen_Connect_Graphs;

{Generates graphs of user-specified connectivity on minimum}
{number of edges on a user-specified number of vertices.   }
{By Dave Carson and Stuart Melville.              July 1991.}

USES
  Printer,
  Crt,
  Library;

CONST
  Random_Base_Hour = 6;
  Mean_Delay = 20;
  Std_Dev_Delay = 14;
  Max_Vertices = 100;

VAR
  Output_to_a_File,
  Output_to_the_Screen,
  Using_Printer     : BOOLEAN;
  Adj_Matrix        : ARRAY [1..Max_Vertices, 1..Max_Vertices] OF 0..1;
  Num_Vertices,
  Num_Edges,
  Max_Connect_Req   : INTEGER;

{-------------------------------------------------------------------------}
PROCEDURE Initialise;

VAR
  a_Char : CHAR;
  Row : INTEGER;

BEGIN
  Clrscr;
  WRITE ('Please enter number of vertices in the graph : ');
  READLN (Num_Vertices);
  WRITE ('Please enter connectivity of the graphs from 2 to ');
  READLN (Max_Connect_Req);
  WRITE ('Output to the printer? ');
  Achar:= Get_Valid_Choice_From( ['Y','N']);
  WRITELN;
  Using_Printer := (a_Char = 'Y');
  WRITE ('Output to a file? ');
  Achar:= Get_Valid_Choice_From( ['Y','N']);
  WRITELN;
  Output_to_a_File := (a_Char = 'Y');
  WRITE ('Output to the Screen?');
  Achar:= Get_Valid_Choice_From( ['Y','N']);
  WRITELN;
  Output_to_the_Screen := (a_Char = 'Y');
  Fillchar (Adj_Matrix, Sizeof(Adj_Matrix), 0);
  Num_Edges := 0;
  Clrscr
END;
{-------------------------------------------------------------------------}
```

```
{-------------------------------------------------------------------------}
PROCEDURE Display_Matrix;

VAR
  Row, Col : INTEGER;

BEGIN
  WRITELN;
  WRITELN ('Displaying Matrix...');
  WRITELN ('Number of edges is now ', Num_Edges);
  FOR Row := 1 TO Num_Vertices DO
    BEGIN
      FOR Col := 1 TO Num_Vertices DO
        WRITE (Adj_Matrix [Row, Col] : 3);
      WRITELN;
    END;
  READLN;
  WRITELN
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Display_Matrix_Printer;

VAR
  Row, Col : INTEGER;

BEGIN
  WRITELN(lst);
  WRITELN (lst,'Displaying Matrix...');
  FOR Row := 1 TO Num_Vertices DO
    BEGIN
      FOR Col := 1 TO Num_Vertices DO
        WRITE (lst,Adj_Matrix [Row, Col] : 3);
      WRITELN(lst);
    END;
  WRITELN(lst)
END;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
PROCEDURE Create_Graphs;

VAR
  Loop,
  Connect_Reqd : INTEGER;
```

```
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Augment_Graph;

VAR
  Current, Partner, Loop : INTEGER;

BEGIN
  IF (NOT Using_Printer) AND (Output_to_the_Screen)
    THEN clrscr;
  WRITELN ('Augmenting Initial Graph to Connectivity ', Connect_Reqd, ' ...');
  Current := 1;
  REPEAT
    Partner := Current + Connect_Reqd DIV 2;
    IF Partner > Num_Vertices
      THEN Partner := Partner - Num_Vertices;
    IF Adj_Matrix [Current, Partner] = 0
      THEN BEGIN
        Num_Edges := Num_Edges + 1;
        Adj_Matrix [Current, Partner] := 1
      END;
    IF Adj_Matrix [Partner, Current] = 0
      THEN BEGIN
        Num_Edges := Num_Edges + 1;
        Adj_Matrix [Partner, Current] := 1
      END;
    Current := Current + 1
  UNTIL Current > Num_Vertices;
  IF Using_Printer
    THEN Display_Matrix_Printer
    ELSE IF Output_to_the_Screen
           THEN Display_Matrix
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
PROCEDURE Write_Graph;

VAR
  Row,
  Col             : INTEGER;
  Out             : TEXT;
  Connect_Reqd_Str : string;

BEGIN
  WRITELN ('  Writing Data ...');
  STR (Connect_Reqd, Connect_Reqd_Str);
  ASSIGN (Out, 'Conn' + Connect_Reqd_Str);
  REWRITE (Out);
  WRITELN (Out, Num_Vertices);
  WRITELN (Out, Num_Edges);
  FOR Row := 1 TO Num_Vertices DO
    FOR Col := 1 TO Num_Vertices DO
      IF Adj_Matrix [Row, Col] <> 0
        THEN WRITELN (Out, Row:4, Col:4,
                           Mean_Delay:4, Std_Dev_Delay:4);
  CLOSE (Out)
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


BEGIN
  FOR Loop := 1 TO Max_Connect_Req DIV 2 DO
    BEGIN
      Connect_Reqd := 2 * Loop;
      Augment_Graph;
      IF Output_to_a_File
        THEN Write_Graph
    END
END;
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
```

```
BEGIN                    {Main Program}
  Initialise;
  Create_Graphs
END.
```