NEWSLETTER

EUUG

European UNIX® systems User Group

Volume 7, No. 2
1987

CONTENTS

- The Native Language System
- Grouse: Messages and Prompts
- ANSI/ISO C Standardisation
- News Scheme Proposal
- Printing Music Scores

EUROPEAN UNIX SYSTEMS USER GROUP NEWSLETTER

Volume 7
Number 2

Editorial 1

Packets Vs. Circuits, in Two Centuries 3

Music: a Troff Preprocessor for printing music scores 7

An Overview of the Native Language System 25

Grouse: Messages and Prompts in Programs 35

Another Proposal for a News Scheme 45

UKUUG 1987 Summer Technical Meeting 47

Glasgow Local UNIX Group 49

EUUG 51

Progress of ANSI/ISO C Standardisation 53

X/OPEN — What, Who, Why, When 59

EUUG Tape Distributions 61

EUnet 67

UNIX Clinic 71

Review of POSIX 73

This newsletter was set in Times Roman on a LaserGrafix LG800 printer, driven by troff from the UNIX time-sharing system.

Printed by:	Atterbury Associates 2 Passey Place London SE9 1BN U.K.
Published by:	The EUUG Owles Hall, Buntingford Essex SG9 9PL U.K.

The EUUG Newsletter: Time for Another Conference.

Alain Williams
...!mcvax!ukc!insetl!phcompl!addw

EUUG Newsletter Editor

1. *Total Chaos*

I sit typing this in the midst of preparing to fly to Helsinki. An old envelope lists items not to be forgotten: ticket, passport, socks, sea sickness pills, ... A pile of other necessities is growing in the middle of the floor; my new suitcase is starting to look a bit small. How should I pack the wine so that Customs don't notice that I have too much?

By the time you read this I will have nursed a sore head on board the M/S Mariella, as I traditionally do on the second morning of an EUUG conference. Some of the papers look interesting (add a notepad to the list), I will get the best published in the next newsletter. I hope that they serve better coffee than at Manchester.

Where did those pills go to?

2. *UNIX Soaps*

This issue sees the birth of regular features in the newsletter. These are the first of a series of articles by the same person with some particular theme. Their nature varies: in many cases it will keep you up to date with developments in the field from the point of view of the author; in the others different topics will be discussed in some related way.

There are more in the pipeline than have met today's copy date. Let me know what you think of them, and what new themes you would like to see.

3. *The Next Newsletter*

The copy date for the next issue is the 1st of August. All sorts of articles are needed: short, long, serious, or funny. Too many of the articles in this issue are written in the UK: now it's time for *you* to show that *you* exist and have done something interesting.

If you are organising an event or wish to announce a new product, or your marriage: mail me the details and I will put it in as a *short*.

If you have good photographs we can print them — send me one of your user group chairman at his Christmas party.

You may e-mail articles to me, or to euug-n@mcvax. If you want to chat, you can reach me on +44 1 435 0200. I will send rough guidelines on how to produce a paper to anyone that asks.

EUUG

European UNIX® systems User Group

Are YOU a UNIX® User?

If so, you can dramatically extend your knowledge of UNIX systems and share your views and expertise with other UNIX users throughout the world by becoming a Member of the

European UNIX systems User Group

The EUUG is a non-commercial organisation which is in-being to:

* **Enable regular contact between UNIX users**
* **Provide a source of unbiased information**
* **Provide forums for discussion by means of exhibitions, conferences and symposia**
* **Improve knowledge of UNIX and thus improve the System itself**

In addition to a quarterly **Newsletter,** the EUUG makes available its EUnet **electronic mail** and news services to all Members; supports the activities of its National User Groups throughout Europe; provides a **Software Distribution** Service; and has a large number of specialists within its Membership to help users with their problems.

Its International Conferences are now a very important part of the UNIX scene with future events scheduled to take place in Finland and Sweden (May 1987); Dublin (Autumn 1987); and London (Spring 1988).

If you want to play a bigger part in promoting the cause of the UNIX Operating System and — at the same time — benefit from the many services offered by the EUUG, then contact the Secretariat NOW for further details.

European UNIX® systems User Group
Owles Hall, Buntingford, Herts SG9 9PL, England Tel: 0763 73039 Network address: euug@inset.uucp

Packets vs. Circuits, in Two Centuries

Michael Lesk
lesk@cs.ucl.ac.uk

UNKNOWN AFFILIATION

Should complex transmission systems make routing decisions at each hop of each item, or should routing decisions be made in advance for an entire transmission? Recently this argument has been best known as a dispute between packet switched and virtual circuit computer data networks. In the last century, the railroads had a similar problem: should trains proceed signalman by signalman, or according to complete schedules? If the analogy really holds, heavy traffic networks should prefer virtual circuits.

In the last century, the Midland Railway in England found itself with a problem. Coal wasn't getting through to London. At this time (1890's) they, and all other British railways, scheduled their passenger trains but not their slow freights. A coal train meandered from signal cabin to signal cabin, each signalman deciding whether to send it forward or hold it on a siding. He did that by telegraphing the next signalman forward, and asking whether the line was clear. If the next signalman accepted the train, it moved forward one block section, and then the process repeated.

Readers should understand some of the constraints here. Typical British goods wagons did not then have any kind of brake that could be applied while the train was moving. Until about ten years ago, on British freight trains brakes were often found only on the locomotive, tender and guard's van. Even though these trains were short by some standards (British railways were built with short sidings and locomotives were at that time much less powerful than today), they were still underbraked, and safety was preserved by running at slow speeds. So a coal train out on the line represented a substantial delay to any passenger train caught behind it.

Time-keeping on passenger trains, of course, was important. Signalmen were instructed that delaying any passenger train was a serious error, while it did not matter much how fast coal moved down the line. So they tended to be conservative, and when in doubt to put the freight trains into the sidings. But now consider the consequence: suppose you are a signalman, and all your siding space is full of trains, and an adjacent signalman proposes to send you another train; you must refuse it, because if you can not get one of your sidings empty before it arrives a main line will be blocked.

You can now imagine the consequences: as the traffic builds up, the sidings fill, and once all the sidings at important junctions are full nothing can move. The line may be empty; but the trains will sit there until the sidings clear from the destination back. The Midland Railway eventually had to pay enginemen for a day's work in which their train never moved, and it had to refuse traffic because it had no place to put the trains. Computer scientists will recognize this behavior as "thrashing" if too much working memory is needed by the resident processes the CPU spends its time swapping to disk and only rarely does a process get to run.

The normal English solution to the situation was to put down additional tracks. Two-track railways became four-track or six-track. The Midland didn't have the money for that. Instead, a new general manager named Cecil Paget invented the idea of "train paths". Roughly speaking, the idea was that you could accomodate the irregular freight traffic by having a schedule that displayed not only the trains that ran each day, but also showed the slots for additional trains that only ran occasionally. The schedule showed not just the trains that were, but all the trains that might be.

In a train path schedule, the compiler first works out the maximum capacity of the line. This involves picking minimum intervals between trains, and also deciding on the maximum speed trains can be expected to maintain. Since different classes of trains will run at different speeds, the schedule must present a balance between slow freights, passenger expresses, and everything in between. In addition, both long distance and local services must be provided for. And when trains must meet or pass, the schedule must arrange that this happens at a place with a siding.

Once the full list of possible paths was made, the job of the train dispatcher became easier. Given a new request for a slow coal train to London, he simply looked down the list for the next appropriate path, and telegraphed all the signalmen that a train would run on that path today. They could now send it on confidently because they knew that it would not interfere with more important traffic. At each point where there was a conflict, the schedule would so indicate and there would be a siding. As the train left the mine, its arrival time in London would be known.

Doesn't this all sound familiar? The choice is between pre-routing and dynamic routing. Today we have packets or datagrams instead of trains, we have network nodes instead of signalmen, and we have store-and-forward buffers instead of sidings. But we have the same basic problem: in heavy traffic, dynamic routing can overfill buffers and prevent a guarantee of service. Virtual circuits may instead deny a circuit, but once the circuit is allocated the routing is known and the service dependable. Some papers in the literature recommend virtual circuits for heavy traffic (Butrimenko, 1979); others recommend datagrams for heavy traffic (Matsushita, Yamazaki, and Yoshida, 1977).

What is the lesson? If we accept the analogy, we should conclude that

a. virtual circuits should be preferred to datagrams in heavy traffic situations;

b. pre-routing is more efficient than dynamic routing.

Of course, there are differences between railroad scheduling and data switching. The variety of routings in data networks far exceeds that in railroading, and the ratio of data transmitted to storage in the network nodes is greater. But don't be too smug about present day systems: after all, no Victorian railwayman could throw away a train of coal and ask the mineowner to please retransmit.

References

O. S. Nock *Steam Railways in Retrospect* A. & C. Black London, 1966.

Cecil J. Allen *Railways of Today* F. W. Warne London, 1929.

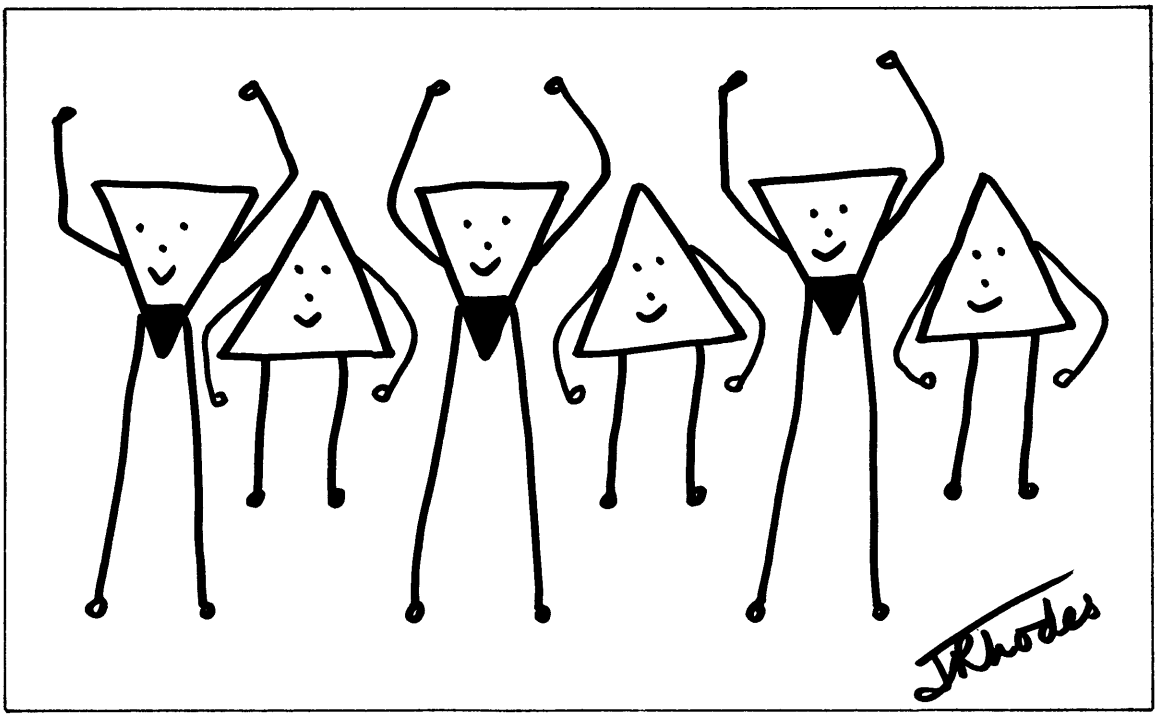
F. J. Salzborn *Timetables for a Suburban Rail Transit System* Transportation Science Volume 3, Number 4, Pages 297-316, 1969.

Yutaka Matsushita, Haruaki Yamazaki, Isamu Yoshida *An Evaluation of Virtual Circuits and Lettergram Services* Computer Networks, Volume 3, Pages 287-294, 1979.

Alexandr Butrimenko, Ulrike Sichra *Virtual Circuits vs. Datagram - Usage of Communication Resources* Performance of Computer Systems, Pages 525-537, North-Holland, 1979.

Vinton G. Cerf, Peter T. Kirstein *Issues in Packet-Network Interconnection* Proc. IEEE, Volume 66, Number 11, Pages 1386-1408, 1978.

G. Pujolle, O. Spaniol *Modelling and Evaluation of Several Internal Network Services* Performance Evaluation, Volume 1, Pages 212-224, 1981.



UNIX Clones

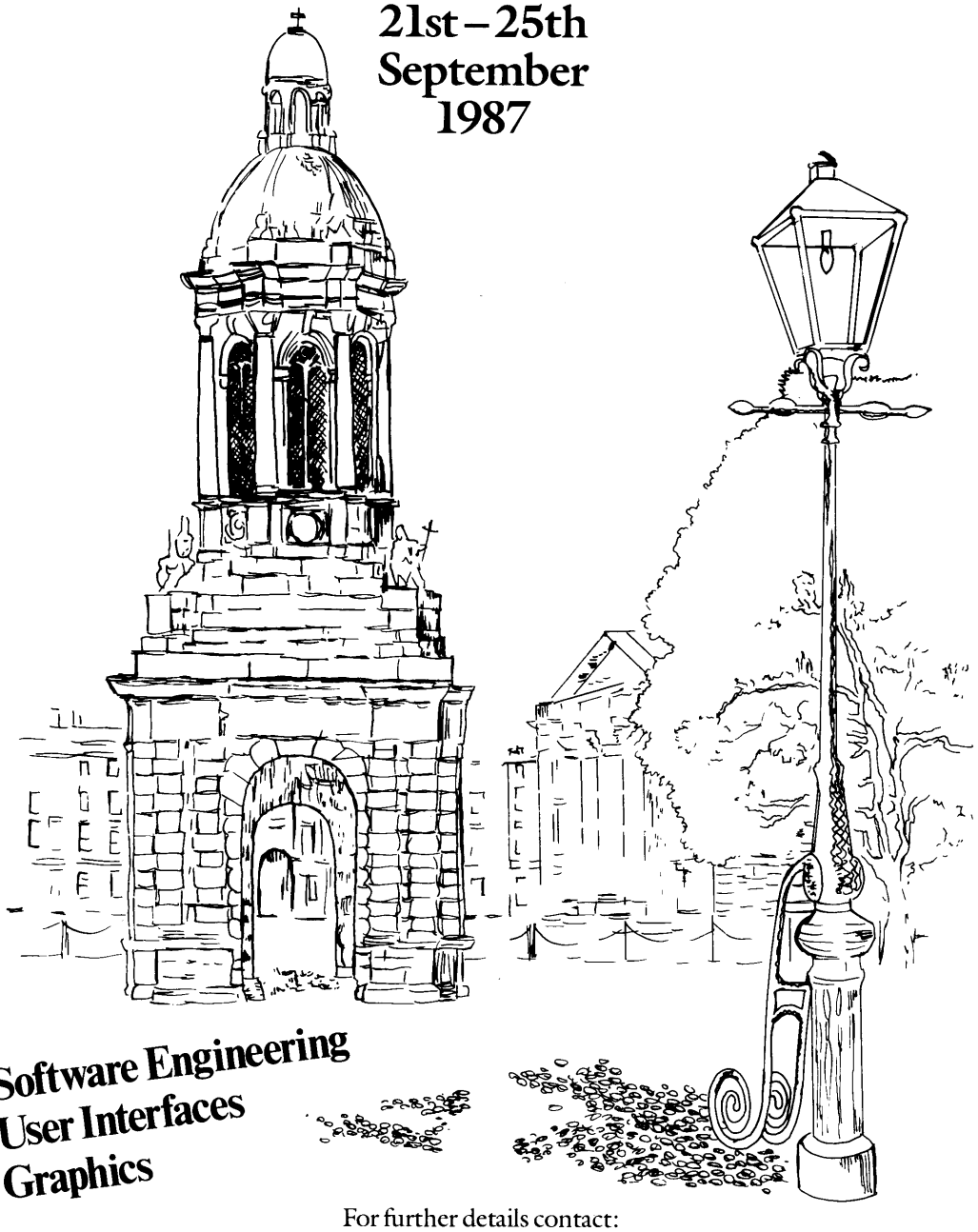
EUUG

European UNIX® systems User Group

AUTUMN MEETING

UNIVERSITY OF DUBLIN
Trinity College

21st – 25th September 1987



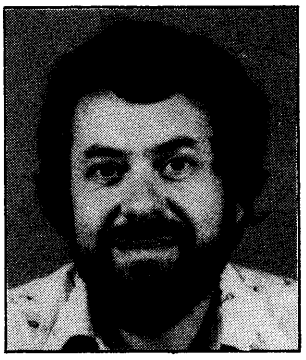
- **Software Engineering**
- **User Interfaces**
- **Graphics**

For further details contact:

The Secretariat: European UNIX® systems User Group,
Owles Hall, Buntingford, Herts SG9 9PL, UK.
Tel: Royston +44 (0) 763 73039 Facs: Royston +44 (0) 763 73255
Network address: euug@inset.uucp

®UNIX is a Registered Trade Mark of AT&T in the USA and other countries.

Music:
A Troff Preprocessor for printing music scores



Eric Foxley
ef@cs.nott.ac.uk

Departments of Mathematics
and Computer Science
University of Nottingham
U.K.

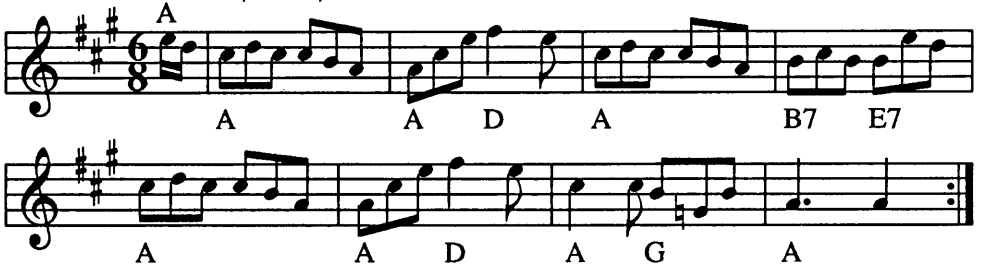
ABSTRACT

The *music* preprocessor provides a language for describing music scores, which can then be processed to produce output suitable for the *troff* typesetting system and its other preprocessors, which run under the UNIX† operating system. This document describes the basic facilities available in the *music* preprocessor, and gives examples of its use.

1. Introduction

The output

Cock of the North *(Kevin)*



was created from an input file containing

```
.MS
title = "\fBCock of the North\fP \fI(Kevin)\fP";
timesig = 6 8;
autobeam;
key = a.

e< 'A' d< |
c "A" d c c b a |
a "A" c e f^> "D" e |
\1 |
b "B7" c b b "E7" e d |
\1, 2 |
c> "A" c b "G" g= b |
a>. "A" a> :|
.ME
```

† UNIX is a trademark of Bell Laboratories.

The *music* system is another *troff*[4,8] preprocessor. It passes most of its input through untouched, but translates those lines between lines ".MS" and ".ME" into commands suitable for the *pic*[5,6] preprocessor, which can then draw the necessary pictures. Text outside and inside the *music* system can use the full features of the other *troff* preprocessors such as *eqn*[3] and *tbl*[7] if required. A typical UNIX command would then be

```
music source.file | pic | troff -ms
```

or

```
music source.file | pic | eqn | troff -ms
```

if the facilities of *eqn* are required.

The particular rules for layout of the musical symbols are based on examples given in Stone[10] and in the Oxford Concise Dictionary of Music[9] subject to interpretation and variation. Suggestions for alternative rules would be appreciated, and could perhaps be built into the system as options. For a more general discussion in this area, the reader is referred to[1] for discussion of languages for representing music scores, and to[2] for information on the current state of comupter applications in music printing and in general musicology.

2. The input language

The basic input for the musical score is contained between lines ".MS" and ".ME", and consists of header information describing the output format required, and input defaults, followed by the score details. All text not within a ".MS" to ".ME" section is passed straight through unchanged.

2.1. Header information

The header sets up variables such as the piece title, output width, time signature and key, each of which is specified by an entry of the form

```
<identifier> = <value>
```

The header items are separated by semi-colons, and terminated by a full-stop. A typical header for a straightforward example might be

```
title = "Twinkle twinkle little star";
timesig = 4 4;
key = d.
```

In all straightforward cases a short header is acceptable, since most items default to sensible values. However, the header items have to allow for many variations in output format, and examples of the major possibilities are shown in the following example:

```
title = "...."; # printed at top left of output; the default is to have no title
ctitle = "....";# this title is printed at the top centre of the output, if given
rtitle = "....";# this title is printed at the top right of the output, if given

timesig = 3 4; # sets the default note length to semi-breve divided by 4,
               # and the default bar length to this times 3
               # the musical length of bars is checked against the bar length

key = g;       # the key of the piece is "G" for both input and output
               # all "F" notes on input now default to F-sharp
keyin = f;     # the input key can be specified distinctly from
keyout = a;    # the output key to produce transposed output
               # the "key =" entry sets both these values
transpose = 1 -1;# instead of using "keyin" and "keyout", this option
```

```
                # specifies the number of additional sharps and octave displacement in the
                # output key compared with the currently set output key

octave = s;     # sets the default octave to that within the stave of the treble clef
                # "octave = c;" causes the default octave to start at middle "C"
                # "octave = k;" causes the default octave to start at the current key note
                # "octave = p;" causes the default octave to make each note as close as
                # possible to the previous; the first note follows the "octave = s" rule

bars = 8;       # the number of bars in the piece
                # used for checking, incorrect value produces a warning
bps = 8;        # bars per stave; bars are spaced on the stave proportionally to their
                # musical length; notes within a bar are spaced
                # proportionally to their musical length
                # bps = 0 (default) fits as many bars as possible on the stave

width = 6.5i;       # the stave width in inches
                # the maximum depends on the output device;
                # the current troff default width is 6 inches
                # our current laser printer maximum is 7.5 inches portrait
                # and 11 inches in landscape mode
height = .25i;  # the height of a 5-bar stave
                # the default is 0.28 inches
isg = .20i;     # increase the spacing to be left between staves,
                # the "inter-stave-gap" by 20 units

sig = ckt.      # the "c" is "print a clef at the left of each stave"
                # the "k" is "print the key on each stave"
                # the "t" is "print the time-signature on the first stave"
                # the default is to have all
```

The header items are in any order, separated by semi-colons; the last is terminated by a full stop. All unspecified items default from any previous header. An empty header is indicated by a full stop. Further header items will be described later.

Note that all text from any "#" symbol to end-of-line is ignored, and that if the last character of a line is the escape character "\", then the next line is treated as a continuation of the current line.

2.2. Score details

The header is terminated by a full stop, and is followed by the score. The score consists of notes interspersed with bar-lines. There is a warning if the sum of the note lengths in any bar does not add up to the required bar length as deduced from the time signature, or if the total number of bars does not agree with that specified in the header. Both of these checks have been found to be useful.

The pitches of the notes are typed as lower-case letters relative to the current key, as in

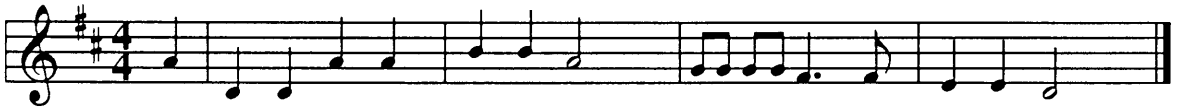
```
d d a a | b b a | g g f f | e e d
```

Sharps and flats of the current key are omitted. Other required accidentals are typed at the first occurrence in the bar using "+" for sharp, "-" for flat, and "=" for natural. For example, "g+" represents g-sharp, "e-" represents e-flat, and "f=" represents f-natural in a key such as "d". A "+" symbol against an already sharpened note is ignored; a "+" symbol against a note which is sharp in this key, but which occurred with a natural earlier in the bar, cancels the natural accidental. On output, the computer will print only the necessary accidentals, omitting, for example, accidental signs on all but the first occurrence of a given accidental within a bar. A cancelling accidental will be printed if the note is used in the following bar.

The length of a note defaults to the value indicated by the denominator of the time signature, and is thus a quaver if the denominator is 8, a crochet if it is 4, and so on. To specify other lengths, symbols ">" after the note double its length, symbols "<" halve it, "." increases it by 50%, and ".." increases it by 75%. Thus in 4/4 time, "c>." represents a dotted minim, and in 6/8 time "c<<" represents a demi-semi-quaver. As an example, the source

```
.MS
timesig = 4 4;
key = d;
bars = 4.
d d a a | b b a> | g< g< g< g< f. f< | e e d> |
.ME
```

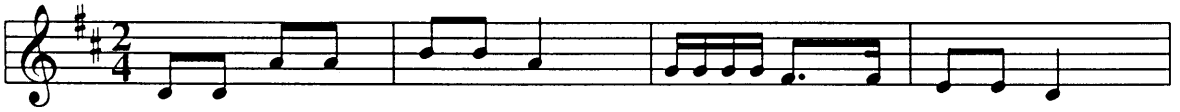
produces the output



The length of the default note for input may be changed by using these note duration symbols in association with the note given as the key in the header. Thus if the key is specified as "d>", the key is "d", and the default note length is double that which would otherwise be expected. If the above example is repeated changing two header entries to halve both the bar-length and the default note-length, the file becomes

```
.MS
timesig = 2 4;
key = d<; # half-length default note
bars = 4.
d d a a | b b a> | g< g< g< g< f. f< | e e d> |
.ME
```

and the resulting output is



For this effect, the key must be set **after** the time signature in the header, since the "timesig" entry itself resets the default note-length. The full-stop cannot be used to increase the default note-length by 50%, the default can be changed only by factors of 2.

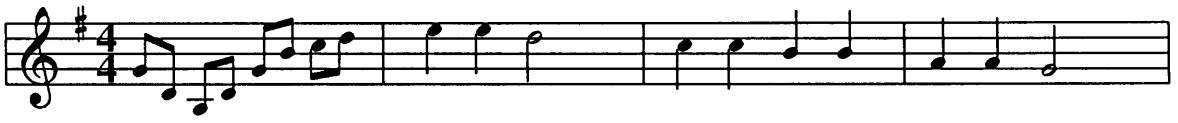
For each note in the source, the letter (and possible accidental) specifying the pitch can be followed by an indication of octave displacement. Symbols "↑" after the pitch indicate an octave upward, symbols "↓" indicate an octave down. The default octave can be set in different ways. It can be set to that from middle C to the B above using the header entry

```
octave = c
```

In this case "c↑↑" is two octaves above middle C, and "b↓" is one note below middle C. Thus in the key of G, the score

```
g< d< b↓< d< g< b< c↑< d↑< | e↑ c↑ d↑> | c↑ c↑ b b | a a g> |
```

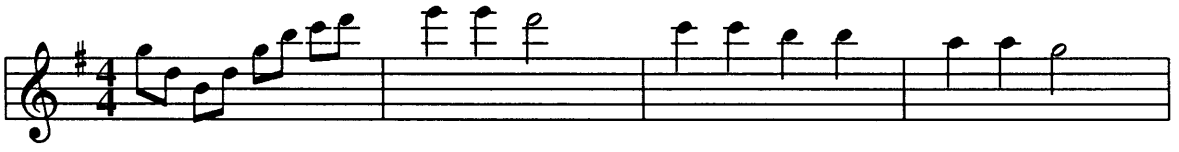
produces the output



The octave symbols(s) must at present precede the note duration symbol(s). The default octave can be moved up or down an octave by appending "↑" or "↓" symbols to the note in the key definition in the header. The previous example, if the default key is specified as

key = g↑; # g up an octave

we obtain the output



The header entry

octave = s;

sets the default octave to that contained within the treble clef stave, from F above middle C to the E above that. The header entry

octave = k;

sets the default octave to be that starting at the current key note. The entry

octave = p

causes the octave of each note to be chosen to make its pitch as close as possible to that of the previous note. Thus in this mode the notes

a b c d l e f g a l a g f e l d c b a l

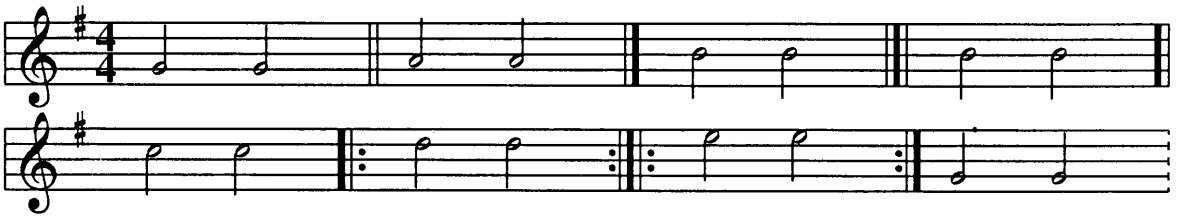
would give a run up and down an octave scale. In this mode, the very first note is set according to the "octave = s" rule. Note that although this option minimises the typing of input, it has the unfortunate side effect that octave errors now propogate throughout the rest of the piece.

Bar-lines

Bars are delimited by bar-lines. A limited variety of bar-lines is available, some indicated by an obvious construction, others by a letter following the bar symbol. A selection of these is indicated by

g> g> ll a> a> lT b> b> lU b> b> lV c↑> c↑> l: d↑> d↑> :l: e↑> e↑> :l g> g> lH

which produces



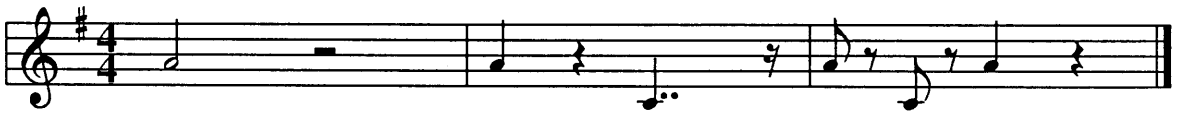
Note that the system is sufficiently intelligent to replace for example the ":l:" symbol if it occurs at the end of a stave by a ":l" symbol at the end of that stave, and a "l:" symbol at the beginning of the next stave.

Rests

Rests are indicated by the underscore symbol "_", with lengths specified as for notes using the symbols ">", "<" and ".". The default length is the same as the value set for notes. Not all rest symbols are currently implemented on the printer. The input

```
a> _> | a _ c.. _<< | a< _< c< _< a _ |
```

gives the output



Beams

To cause notes shorter than crochets to be joined under a common beam, the notes to be joined are put within square brackets, as in

```
[a b c] [d e f]
```

A beam will correctly fail if it includes any notes longer than or equal to a crochet; at the moment, it also fails if it includes any rests. The latter restriction should be removed soon. A beam cannot cross a bar-line. As an example of beams, the source

```
timesig = 4 4; key = d.
d↓ [ d↓<< e↓<< f<< g<< ] a a | [ b< a< g< b< ] a> |
g [ g<. g<< ] f [ f< g< ] | [ a<< g+<< a<< b<< ] [ a<< g<< f<< e↓<< ] d↓> |T
```

appears as

To simplify the typing of input in straightforward cases, a system for automatically inserting beams is available, and can be invoked by inserting the additional line

```
autobeam;
```

in the music heading. The automatic beaming uses rules from the Oxford Concise Dictionary of Music[9] which may not always be exactly what is required. However, automatically generated beams will never over-ride beams inserted manually. Having been invoked, auto-beaming can be switched off by the header

```
autobeam = 0;
```

As a further facility, the auto-beamer can be instructed never to let a beam cross for example a "2 times default note-length" interval using

```
autobeam = 2;
```

Different formulae for the slope of the beam are built into the program; formula number 2 can be accessed by the header entry

```
beamslope = 2;
```

Details are the formulae are available from the author. Other formulae could easily be added; suggestions are welcome.

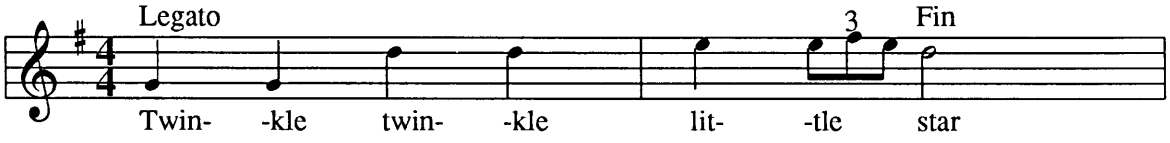
Text

The title of the piece as (and if) given with the keyword *title* in the header information appears at the top left corner of the output. Other titles can be given to appear at the the centre and at the top right-hand corner of the output using the keywords *ctitle* and *rtitle* in the header.

Additional items of text can be given to be positioned relative to any note or bar. For text associated with notes, the text required is contained within quotes to indicate its positioning. Single quotes ('Moderato') indicate text to be positioned above the stave, left justified and starting at the horizontal co-ordinate of the note; text in double quotes ("Twinkle") indicates text to be positioned below the stave; and text between "@" signs (@3@) indicates text to be positioned close to the note. Text of the last form will be positioned just above the note if it has a downward stick, and *vice versa*. Thus the input

```
g 'Legato' "Twin-" g "-kle" d "twin-" d "-kle" |
e "lit-" [ e< "-tle" f↑< @3@ e< ] d> 'Fin' "star" |
```

generates the output



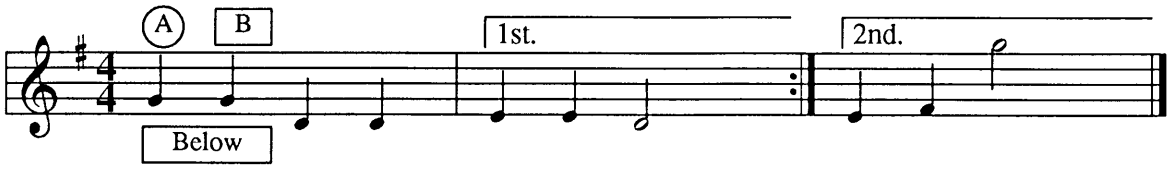
The particular string @3@ is interpreted to imply a triple, three notes in the time of two. The spacing of the notes on the score and the checking of bar-lengths takes this into account.

Text can also be associated with a bar. It is given after the bar-line, and is printed lined up with the start of the bar; it will be above the stave if contained in single quotes, and below if in double quotes.

Text strings starting with the escape character "\" are treated specially. Any text string starting "\ b" appears in a box, any starting "\ c" appears in a circle. and text starting "\ 1" or "\ 2" above the stave is used for alternative bars on repeats. The second type, starting "\ c", will be expected to be single character strings. For example, the file

```
g '\ cA' "\ bBelow" g '\ bB' d d | e '\ 1st.' e d> :| e '\ 2nd.' f g↑> |T
```

produces the output



Text starting "\ >" is assumed to be a *diminuendo*; its width will stretch over two notes if written "\ > 2n", over two bars if written "\ > 2b", and over 2 inches if written "\ > 2i".

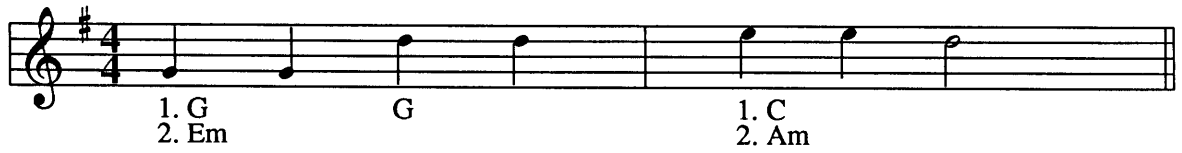
Text between @ symbols starting "\ .", "\ -", "\ <", "\ >" is interpreted as accent symbols and positioned accordingly.

It is hoped that most recognised musical symbols (such as pause and segno) will eventually be included in the standard font; such additional characters are then accessed by the usual methods appropriate to *troff*, such as the "\ (sh" for the sharp-character, and "\ (ft" for the flat-character.

If text is given as two strings, such as

```
"1. G" "2. Em"
```

the two text strings "1. G" and "2. Em" appear one above the other, vertically aligned at the left, as in



Ties

Ties are indicated by putting parentheses (round brackets) around the notes to be joined under the tie. Their production is through the use of splines in *pic*; an angular version is available if splines are not provided.

Duplicate copies of earlier bars

The notation

```
\4 |
```

cause a duplicate of bar number 4* to be inserted at this point. The similar notation

```
\1, 3 |
```

causes bars 1 to 3 to be duplicated at this point. Numbering starts at bar 0 for the lead-in notes, and bar 1 for the first full bar. The notation

```
\-1 |
```

causes a duplicate of the previous bar to be inserted (current position minus one), and

```
\-4, -1 |
```

inserts copies of the preceding four bars. The notation

```
\ 'Legato'+2, 'Legato'+4
```

duplicates source from the bar two beyond the most recent which contains the text 'Legato' to the bar four beyond it. Such a bar must occur earlier in the current ".MS" to ".ME" section. Only complete bars can be copied.

It is unwise to use absolute bar numbers in cases where an extra bar could perhaps be added or deleted at a later stage. Relative bar references (using the -4 or 'Legato' notations) are safer in such circumstances. References to bars not yet encountered are unacceptable.

Changes of signature

The notation

```
\timesig = 3 4.
```

occurring at the start of a bar resets the time signature, default note length and bar length on the fly to a new value, causing the new value of the time signature to be printed on the stave. The input and/or output keys can be reset similarly using, for example

```
\key = g<.
```

resets both input and output key, or

```
\keyout = d.
```

resets only the output key. If such a change alters the pitch of the output key, the new key signature will be printed on the stave at that point; if it is used to change only the default note length or octave on input, no

* This bar number must already have been entered, i.e. we must be currently positioned at bar number 5 or beyond.

key signature is printed.

The notation

\barno = 25.

moves the source to the start of bar 25; if the specified bar number is ahead of the current bar, any intervening bars are filled with rests. If the bar number given is less than the current bar, it is assumed that a further part is being added; see multi-part music below for details.

The notation

\sticks = u.

causes the sticks of all following notes to be forced upwards until cancelled by either

\sticks = d.

to send sticks downwards, or

\sticks = x.

to leave sticks free to move in either direction.

The notation

\bps = 4.

resets the current "bars-per-stave" value to the given integer. This can be used to lay out varying specified numbers of bars on each stave. This can however be achieved by a further facility; if the 'I' separator representing a bar-line is replaced by a "!" symbol, the stave is ended at that point.

Any of the above items can be combined as in a normal header, separated by semi-colons and terminated by a full-stop, as in

\bps = 4; keyout = d.

With the exception of "\ sticks =", the above notations starting with the escape character "\" are not guaranteed except when used at the start of a bar.

Changes of width

If, say, music is being printed at 8 bars per stave, but there are 4 bars left over at the end, those bars would normally be spread to fit the full width of the page. To reduce their width, use the inserted header

\width = 400 .

which will decrease the width of the page to 4 inches. To reset back to the original width, use

\width = 0.

Source from a separate file

The ".MS" line can be replaced by

.MS < filename

causing music input to be taken from the named file. The file should start and end with ".MS" and ".ME" lines respectively. This enables music source to be easily tested before insertion into the main text. The line can be extended by header items, for example

.MS < filename keyout = b.

Any information given on the ".MS < file" line and following the filename is read **after** the first heading of the the file which is being read. Thus if the named file contains a piece of music which is required to be printed in several keys, or in several different layouts, this can usually be done using extra header items in

this way. The use of the escape character at the end of a line enables several header items to be given, as in

```
.MS < file  keyout = g↑; rtitle = "arranged Foxley"; \
bps = 4.
```

which specifies a new output key, right-hand title and bars-per-stave setting.

Output key

The facility to specify the output key independently of the input key takes account of the fact that a natural in one key may become a sharp in another. A piece may be printed in a key other than that in which it is entered by using the "keyout = ..." facility in the heading of the piece. To print a particular piece in two distinct keys, first in the key in which it was input, and then in a second key, store the input

```
.MS
rtitle = "\ fIAs input in key of A\ fP";
key = a; chords; autobeam; bars = 8.

e< d< |
c^ "A" a a b< c< | d "E7" b b e< d< |
c "A" a a b< c< | d< "G" c=< b< a< g= e< d< |
c "A" a a b< c< | d "D" b b "E7" e< d< |
c "A" a< c< b "G" g=< b< | a> "A" |
.ME
```

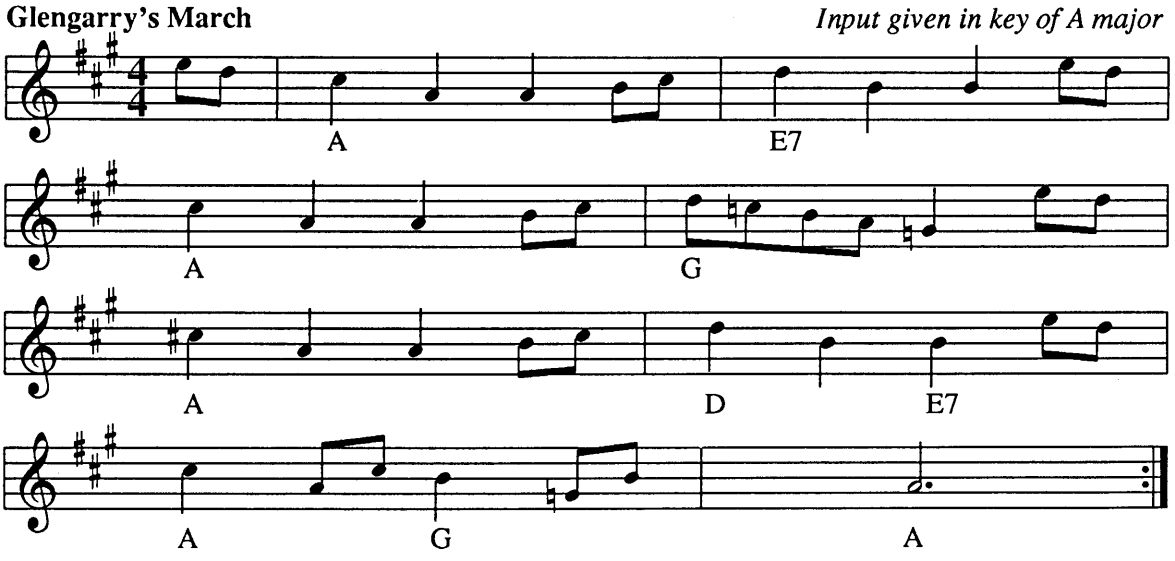
in a file, then use

```
.MS < file
```

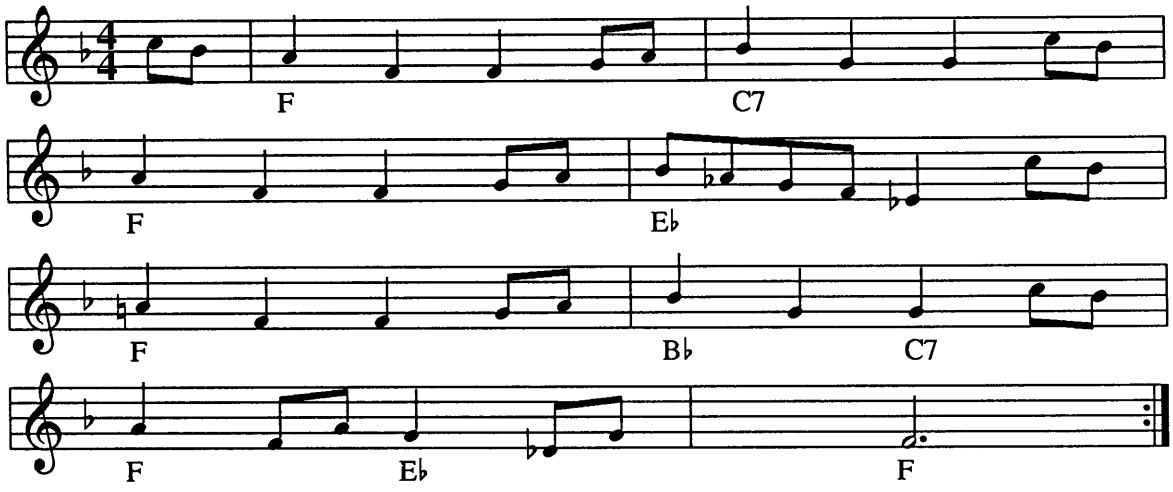
to produce the first copy, and

```
.MS < file rtitle = "\ fIOutput in key of F\ fP"; keyout = f.
```

for a second copy. This then appears as



Glengarry's March *Output in key of F*



An alternative technique for transposition is to use a simple

```
key = d;
```

header, and to follow it with a header entry such as

```
transpose = 2;
```

This will cause all output to be printed two sharps up from the output key currently set, following all output key changes during the piece. Negative values, of course, imply less sharps or more flats.

In transpositions, single sharps, naturals and flats are printed correctly relative to the new key. However, double sharps and flats (arising from, for example, an accidental sharp when transposed to a key in which that note is already sharpened) are re-coded to a new note. It should be noted that the input notation does not currently allow double sharps or flats to be specified in the input key. Both these and quartertones will be added to the program.

The additional entry

```
chords;
```

in the header of the above example causes the system to assume that any text below the stave represents the names of chords, and transposes them correctly; other text is left unchanged. In chord names, it is assumed that the full note is given, for example "F+m" for a chord based on F-sharp, even in a key in which the note of F is by default already sharpened. This appears to be the general convention.

The "chords;" entry also allows the use of "+" to represent sharps and "-" to represent flats in the text of chord names; the "+" and "-" characters will be correctly replaced by "♯" and "♭" symbols when printed. To print a "+" symbol, escape it with the "\" character.

If the "chords" entry is not given, text is printed exactly as specified, so that a "+" sign in the text appears on the output as a "+" sign; a sharp sign would be printed by the string "\(sh".

3. Multi-staff multi-part music

For multi-staff multi-part music, the music source must specify to which part each note belongs, and the parts must then be linked to particular staves.

3.1. Specification of stave layout

Two additional lines must be included in the header, typically

```
staves = t b;
parts = 1 u 1 d 2 x;
```

These indicate

a) that there are to be two staves, the first in the treble clef, the second in the bass clef; and

b) that there will be three parts, the first to be printed on stave 1, sticks up, the second on stave 1, sticks down, and the third on stave 2, sticks either way.* The default is

```
staves = t;
parts = 1 x;
```

To allow for more complicated situations, more information can appear optionally between the clef names on the "staves =" line. If a "=" sign appears between two clef characters, as in

```
staves = t = t b = b;
```

the bar-lines of those two staves will be connected on the final output; in this example the first and second staves will be connected, as will be the third and fourth. If the clef letter is followed by a positive or negative integer, the output on that stave will be in a key offset from the basic output key; a positive integer represents a number of additional sharps or less flats, a negative integer represents a number of less sharps or more flats. This facility is essential for scores involving transposing instruments. In addition this positive or negative key offset may be followed by a number of "↑" or "↓" symbols to indicate octave displacements for that stave relative to the current key. Finally, any string in double quotes following the clef letter will appear at the left of that staff in the output; this is used typically to indicate instrumental parts. The example

```
key = g;
staves = t "flute" t "violin" t "clarinet" +2 t "trombone" -2;
```

will give output on four staves, the first two in the key of G (but perhaps changing during the piece), the third always transposed two sharps up (initially in A), and the fourth always transposed two flats down (initially in F). Permitted clef letters are "t" for the treble clef, "b" for the bass clef, "a" for the alto clef, and "n" for the tenor clef.

3.2. Specifying the separate parts

The music source needs additional notation to identify which of the source belongs to each part. Parts may be given bar-by-bar, as in the two-part music

```
\partno = 1. g g d d \partno = 2. d↓ d↓ b b |
\partno = 1. e e d> \partno = 2. c c b> |
\partno = 1. c c b b \partno = 2. a a g g |
\partno = 1. a a g>      \partno = 2. f f d↓> |
```

When a new part is specified, as in "\partno = 2.", the following source is assumed to restart at the beginning of the current bar. Alternatively, the complete source may be specified part by part, as in

```
\partno = 1. g g d d | e e d> | c c b b | a a g> |
\barno = 1; partno = 2. d↓ d↓ b b | c c b> | a a g g | f f d↓> |
```

where the "\barno = 1" indicates a return to (the beginning of) bar number one.

In multi-part scores, when earlier bars are to be copied, the notation

```
\1, 8 |
```

inserts the current part number from bars one to eight at the current point. The notation

```
\1, 8 p 2 |
```

inserts bars one to eight of part 2 into the current part at the current point. To ignore a particular part which you wish to leave in the file for other reasons, specify it as printing on stave number 0, as in

* The notation "\sticks = u" in the source over-rides the default stave settings.

```
staves = t b;
parts = 1 u 1 d 0 x 2 x;
```

to print the first two parts on stave 1, the fourth part on stave 2, and to ignore the third part.

Facilities to refer to parts by names rather than numbers will be added shortly, together with a transposing option to insert transposed or inverted duplicates of earlier bars.

3.3. Example

Typical source to print a multi-part piece, and then to print the first part on its own, and then to print the last part on its own, would be done by storing the basic score in a file as follows:

```
.MS
title = "Complete score";
key = g;
...
bars = 16;

staves = t b; # treble and bass clefs
parts = 1 u 1 d 2 u 2 d. # two parts on each staff

# source for all parts follows
\partno = 1. ... # soprano part inserted here

\partno = 2; barno = 1. ... # alto part inserted here

\partno = 3; barno = 1. ... # tenor part inserted here

\partno = 4; barno = 1; key = gv. ... # bass part defaults an octave down

.ME
```

To print the complete piece as specified in its header (two staves), use

```
.MS < file
```

To print the soprano part only, use

```
.MS < file title = "soprano part"; # all settings remain as specified \
staves = t; # one staff, treble clef \
parts = 1 x 0 x 0 x 0 x. # one part, sticks either up or down \
```

To print the bass part, use

```
.MS < file title = "bass part"; \
staves = b; # one staff, bass clef \
parts =  0 x 0 x 0 x 1 x. # one part, sticks either up or down
```

Alternatively the full score could be printed on its own, and then the soprano part could be printed alone in a separate run by altering the "stave" and "parts" entries in the first heading to read

```
staves = t;
parts = 1 x 0 x 0 x 0 x;
```

i.e. to ignore parts 2, 3 and 4.

To print on four separate staves, but with the second stave always transposed to two more sharps than the current output key, and the third to one less, the relevant header entries would be

```
staves = t t+2 b-1 b;
parts = 1 x 2 x 3 x 4 x;
```

4. Error reporting

A number of suspicious constructs in the source file produce warning or error messages. After an error message, the program will terminate. After a warning message, processing proceeds as normal. Error messages include the line number of the suspect line, and a pointer to the position at which the error became detected. Possible errors include invalid characters at any point and beams which are opened but not closed. A warning message is produced if the notes in a bar do not add up to the correct length for that bar, but is suppressed if the bar starts or ends with a non-standard bar-line. Un-necessarily repeated accidentals also produce a warning.

5. Miscellaneous additional features

Text omission

The header line

```
text = o;
```

causes text under the stave (in double quotes) to be suppressed, text over the stave printed. The line

```
text = u;
```

causes the text over the stave (in single quotes) to be suppressed, while

```
text = ou;
```

causes both to be printed, and

```
text;
```

causes both to be suppressed.

Cancelled Accidentals

The header entry

```
natural;
```

causes an appropriate accidental to be placed against a note which was set with an accidental in the preceding bar. The entry

```
natural = 0;
```

cancels the effect.

Inter-stave gap

To increase the vertical spacing between staves, use the header entry

```
isg = 25;
```

for an inter-stave gap of 0.25 inches. To increase in addition the gap between the combined staves in multi-staff music, use

```
isg = 25 10;
```

where the first number is the spacing between groups of stave, and the second between staves in a group.

Setting defaults

The program first reads a file called "*mus_default*" if such a file exists. It must be a file containing only a header, of the form

```
.MS
sig = kt;
octave = s;
rtitle = "Copyright Fred's Music Ltd";
height = 0.35i;
width = 700.
.ME
```

and sets all defaults for the program.

PIC commands

The output of the *music* preprocessor is fed into the *pic* preprocessor. Any text string starting "\p...." is assumed to be a PIC command. The program moves to the appropriate position (above the stave if the string is within single quotes, for example) and plants the given PIC commands. In addition, the program reads a file named "*pic_default*" on start-up, to enable PIC-macros to be used.

6. Further enhancements under consideration

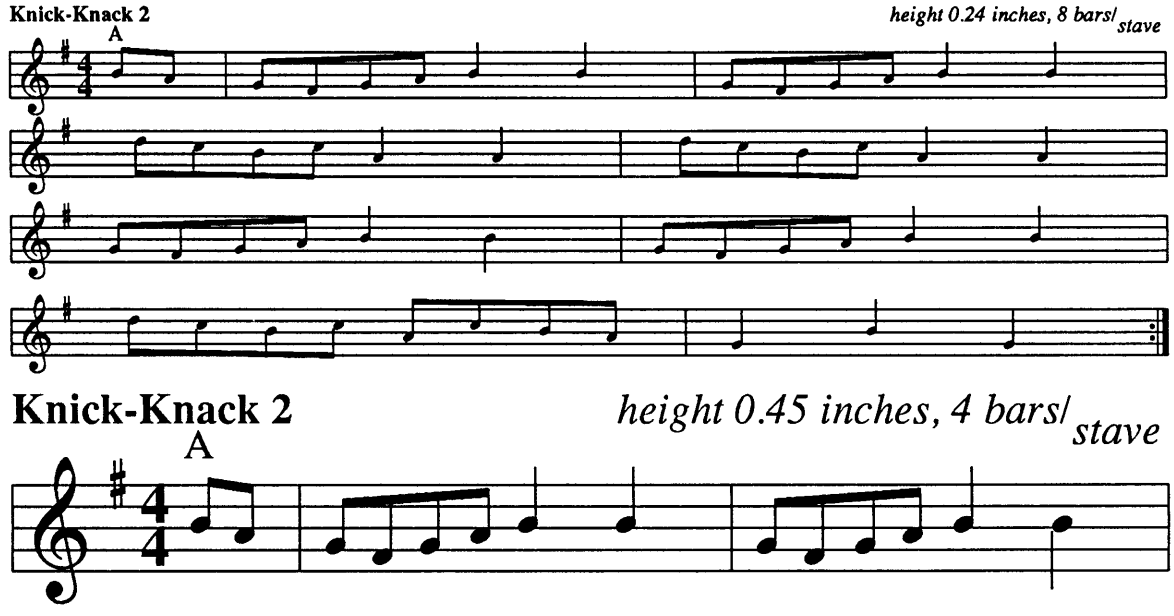
Possible further enhancements include the following.

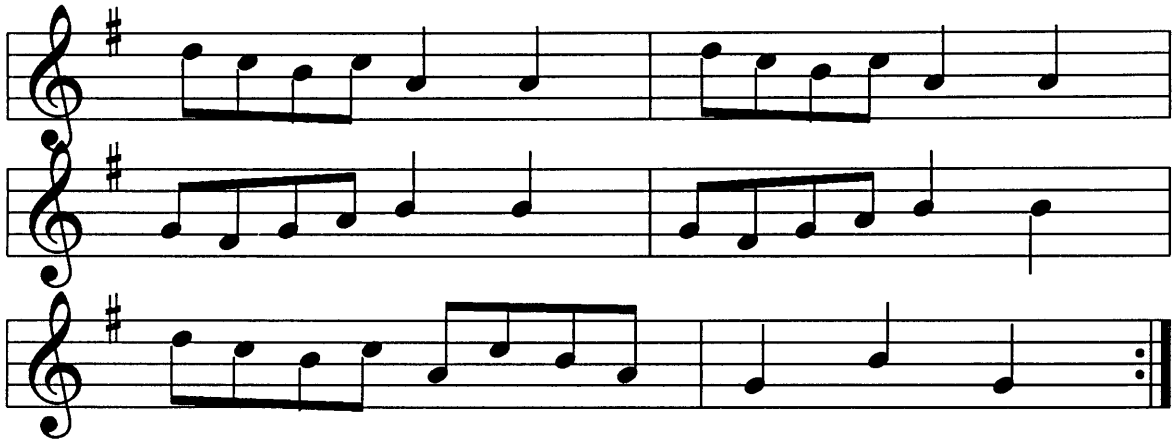
Font characters

The font characters need to be enhanced, new characters added, and a variety of fonts for music symbols made available. Different available digitised music fonts are being studied. The exact alignment of all symbols awaits the choice of a font. (A variety of fonts for text is already available.)

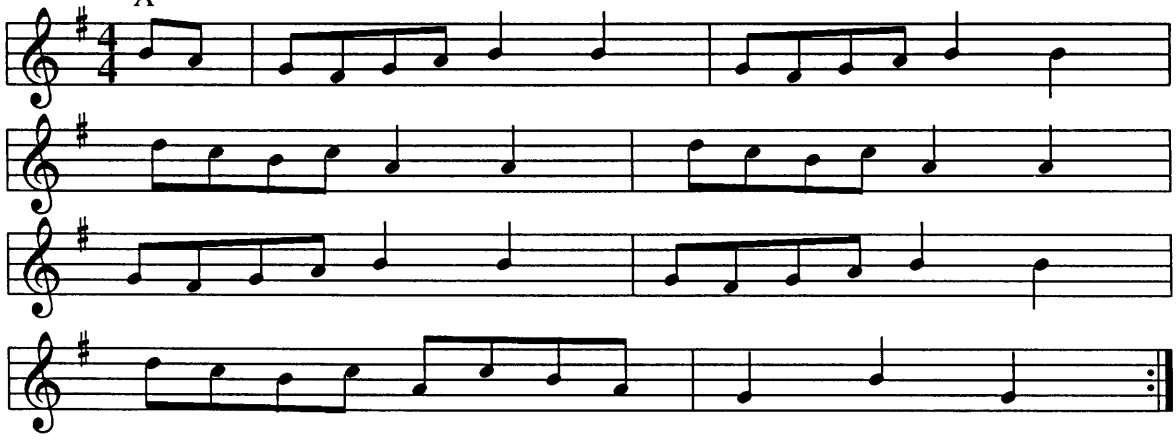
Varying the height of the stave

Refinement of the ability to vary the height of the stave; at the moment some slight misalignment occurs, but the output is as follows:





Knick-Knack 2 *default height 0.32 inches, 4 bars/stave*



Text can, of course, be set to any size using *troff* conventions; the above examples show how the default text size is set in proportion to the stave height.

7. Acknowledgments

Thanks are due to many people for their comments at various stages of the development. The author is indebted in particular to Graeme Lunt who looks after the laser printer, and to Dave Brailsford who obtained it for the department. Other colleagues in my department helped at many points, among them Andy Walker, Jim Duckworth and William Shu. Members of the music department are always helpful, and compensate for my lack of musical expertise; they include Ian Bent, John Morehen and Peter Neslon. The testing of the system has been assisted by Toby Bennett of the Genetics Department, who also devised a system for proof-reading my scores by by writing a program to play them directly on a BBC micro-computer.

Eric Foxley
April 24, 1987

References

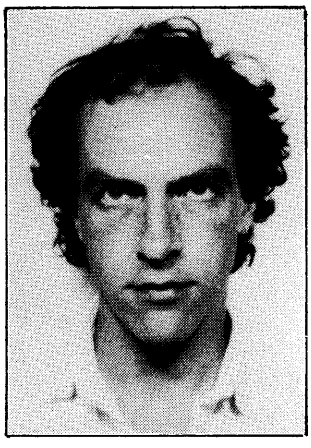
1. John S Gourlay, "A Language for Music Printing," *CACM*, vol. 29, no. 5, p. 37, May 1986.
2. Walter B Hewlett and Eleanor Selfridge-Field, *Directory of Computer Assisted Research in Musicology*, June 1986. Center for Computer Assisted Research in the Humanities, Menlo Park
3. Brian W Kernighan and Lorinda L Cherry, "A System for Typesetting Mathematics," *Comm A C M*, vol. 18, no. 3, pp. 151-157, 1975.

4. Brian W Kernighan, *A TROFF Tutorial*, 1978. Bell Laboratories
5. Brian W Kernighan, "PIC — A Language for Typesetting Graphics," *Software — Practice and Experience*, vol. 12, no. 1, pp. 1-21, 1982.
6. Brian W Kernighan, *PIC — A Graphics Language for Typesetting : User Manual*, March 1982. Bell Laboratories
7. M E Lesk, "Tbl — A Program to Format Tables," *UNIX Programmer's Manual*, vol. 2, January 1979. Section 10
8. Joseph F Osanna, "NROFF/TROFF User's Manual," *UNIX Programmer's Manual 2*, January 1979. Bell Laboratories
9. Percy A Scholes, *The Concise Oxford Dictionary of Music (2nd Ed)*, 1964. Oxford University Press
10. Kurt Stone, *Music Notation in the Twentieth Century*, 1980. W W Norton & Co

An Overview of the Native Language System

Michael J. C. Terry
mjct@inset.co.uk
...!mcvax!ukc!inset!mjct

The Instruction Set Ltd
City House, City Road
London EC1V 9QH



Michael Terry is a technical consultant at The Instruction Set, involved mainly with relational databases.

He originally studied French and Swedish (with a dash of Norwegian and Finnish) at university, before crossing the great divide and plunging headlong into computing ("because there was nothing left to do").

Having an intimate knowledge of European collating sequences, hospitality, and drinking habits, he is uniquely qualified to be working with NLS. Earlier this year he spent 6 weeks at Hewlett-Packard's Cupertino office working with them on their NLS project and learning how to operate the jacuzzi.

"If English was good enough for Jesus, it's good enough for me."

1. A New UNIX Internationalisation Standard

In January this year, the X/OPEN group† published the second edition of its X/OPEN Portability Guide (XPG)[1]. Section 3 of the guide included a software internationalisation interface standard specification — the Native Language System (NLS). Although many proprietary solutions to the internationalisation problem have been attempted over the years, this is the first time that a commercial standard has been specified for internationalisation on UNIX® systems.

The X/OPEN NLS standard specification has arrived as a response to a pressure that has been growing slowly but relentlessly from non-English-speaking UNIX users as use of the system has filtered down from the ivory towers of academe to the air-conditioned offices of modern commerce. It is not surprising that this internationalisation specification has emerged from the X/OPEN group rather than from AT&T — after all, despite the recent addition of American companies to the X/OPEN roll call, X/OPEN started out as a purely European grouping, and is still

† AT&T, Bull, DEC, Ericsson, Hewlett-Packard, ICL, Nixdorf, Olivetti, Phillips, Siemens, Unisys

predominantly European. What is perhaps surprising is that the NLS specification is based on an internationalisation architecture developed in the USA by Hewlett-Packard.

Being Europeans, the members of the EUUG will be well aware of the problems that result from the American nature of UNIX — the ASCII codeset does not support many of the characters in the various European alphabets; the system uses US cultural practices (the US assumptions, for example, that the radix character is a dot and that thousands are separated by commas, are reversed in many European countries); moreover, the terse UNIX error messages are all in (sometimes bizarre) English.

The aim of NLS, then, is to provide the specification of an internationalisation framework — a set of utilites and library functions — that enables applications software to be adapted appropriately for use in *any* country or local environment.

2. *Major Design Goals of NLS*

NLS is designed to provide the following features:

- Support of multiple extended (8-bit) character sets on the same machine, simultaneously.
- Preservation of 8-bit data integrity.
- Multilingual program messages.
- Proper representation of local conventions.
- No need for multiple versions of software for different languages.
- Ability to add new languages without the need to recompile existing software.

In addition, UNIX systems with NLS must still retain the original 7-bit ASCII functionality.

3. *Implementing NLS*

Hewlett-Packard have a working version of NLS on their HP-UX operating system. The source code has been made available to the other members of X/OPEN in order to expedite its implementation on currently available versions of UNIX. Whether they are using the H-P code or not, the other members of X/OPEN are in the process of implementing NLS on their versions of UNIX.

Thus AT&T, having recently joined the X/OPEN group, are committed to the eventual release of NLS on a future version of UNIX. However, it is unknown if AT&T will ever release NLS on any version of System V.2.

Currently, implementing NLS on a UNIX system entails the creation of new utilities and C library functions, as well as modifications to pre-existing UNIX code. No kernel level changes are required.

The next six sections outline the work involved in implementing NLS on UNIX.

3.1 *Extended Character Set Support*

Because ASCII is a 7-bit codeset, it is capable of representing a maximum of 128 characters. 8-bit codesets can represent up to 256 characters.

The eventual intention is that NLS will support multiple 8-bit character sets. The XPG states:

> This first issue of the X/OPEN NLS specification defines the major transmission codeset for Western European use as the standard IS8859/1, and also recommends its use as the corresponding internal codeset. Other codesets will be identified in later issues.

The IS8859/1 codeset is capable of supporting most major Western European languages. In addition, it is compatible with ASCII functionality, since it incorporates the ASCII codeset as the first 128 characters of the codeset.

3.2 Cleaning Up 8th Bit Usage

Everything sounds hunky dory until we take into account the problem that the UNIX utilities have a bad 8th bit habit — many of them use the 8th bit for their own arcane internal purposes. For example, the shell sets the 8th bit of characters read in from the command line if they were quoted. Consequently, 8-bit data is corrupted if passed through any such offending commands.

Another problem is that sign-extension can occur with 8-bit characters when they are manipulated as integers.

The upshot of this is that the UNIX utilities must be gone through with a fine tooth-comb in order to detect and correct any possible areas of corruption. This is a tedious, long-winded and, in the case of some commands, non-trivial task.

3.3 Character and String Handling

Many languages share the same codeset, but there are differences in the way each language handles the component characters of its alphabet.

Lookup tables must be supplied indicating character class membership, up/downshift relationships, and collation (sorting) orders for each language. These tables must understand 1-2 character mappings such as Spanish *ch* and *ll*.

The affected library routines include the `ctype`(3C), `conv`(3C) and `string`(3C) routines, which must be amended to make use of these tables.

3.4 Message Catalogues

NLS uses a message catalogue mechanism to provide program messages and prompts in multiple languages. A utility, `gencat`(1), is used to generate message catalogues from a source file containing program message strings. The library functions `catopen`(3C) and `catclose`(3C) are provided to open and close appropriate message catalogues. The routines `catgets`(3C) and `catgetmsg`(3C) are provided for access to messages from the currently open catalogue.

A message number is associated with each message. This number is used to index into the catalogue to retrieve the message.

An environment variable, NLSPATH, can be used to specify a catalogue search path. The advantage of this is that new languages can be added to a system without the need to recompile software. For example:

```
NLSPATH=/usr/lib/nls/%L/%N.cat:/usr/me/cats/%L/%N.cat
```

The special notation `%L` maps to the name of whichever language is currently being used, while `%N` maps to the name of the program being run. Thus if the current

language is *french*, and the command is `rm`, the above maps to:

```
NLSPATH=/usr/lib/nls/french/rm.cat:/usr/me/cats/french/rm.cat
```

If the catalogue cannot be accessed, a default string (specified in the original source code) is printed out. However, if the *catalogue* is opened succesfully, but the numbered *message* cannot be found, then a null string, rather than the default, is returned. When such errors occur, this results in strangely silent programs. I think X/OPEN got that one wrong.

3.5 Local Customs Database

A local customs database must be supplied for each language. This contains the names of months and days, currency formats, yes/no strings etc.. A routine, `nl_langinfo`(3C), is supplied for accessing elements from this database.

3.6 Language Announcement Mechanism

Means must be provided to enable users and programs to determine the language to be used. An environment variable, LANG, enables the user to set and reset the language in which prompts and messages from internationalised programs will appear. For example, the following shell commands will force any program that uses the `catopen`(3C) call to access the Italian message catalogues from the directory `/usr/lib/nls/italian`:

```
NLSPATH=/usr/lib/nls/%L/%N.cat
LANG=italian
export NLSPATH LANG
```

The `nl_init`(3C) routine is provided to set up the working environment for the current language. This routine reads in the appropriate character tables and local customs data:

```
nl_init( "italian" );
```

`Nl_init`() can be called more than once by a single program, thus for example permitting the same process to work in multiple languages. An example of a very simple program that prompts for a language name and then runs the `date`(1) command in that language is given in Figure 1.

Note that the `%L` notation can be used to force the `catopen`(3C) routine to use the value of the environment variable LANG to determine the catalogue to be accessed. The `nl_init`(3C) routine, on the other hand, uses the value of its single argument to decide which character tables and local customs database will be used. Thus it is possible, by setting LANG to one language and calling `nl_init`(3C) with a different language as its argument, to have a program putting out prompts in one language (say French) while processing data using the tables for another language (say Norwegian).

```
#include <nl_types.h>
#define NL_SETN 1
char *getenv(), *catgets();

main()
{
        nl_catd nlmsg_fd;
        char newlang[15], envstr[20];

        nlmsg_fd = catopen( "menu", 0 );

        for( ; ; )
        {
                printf( catgets(nlmsg_fd,NL_SETN,1, "Make selection: ") );
                catclose( nlmsg_fd );
                scanf( "%s", newlang );
                nl_init( newlang );
                sprintf( envstr, "LANG=%s", newlang );
                putenv( envstr );
                nlmsg_fd = catopen( "menu", 0 );
                system( "date" );
        }
}
```

Example output:

```
Select Language: french
lun 04 mai 1987 10h32 42
Choisissez la langue: swedish
Mån 04 Maj 1987 10.32.43
Välja språket: english
Mon. 04 May, 1987 10:32:44 AM
```

Figure 1. Program to Run the date(1) Command in Multiple Languages

4. *Handling Mixed Codesets — Software Implications*

The nature of the UNIX operating system poses one insurmountable problem when it comes to handling data from a mixture of character sets. UNIX files are merely streams of bytes, and it is impossible to tell what character set the data in a file is composed of.

Although kludges such as file-naming conventions, or storing information on file contents externally to files in some kind of catalogue, etc., have limited usefulness, there is absolutely no way of determining the contents of pipes. The other problem is that a file might contain data from a mixture of character sets.

Consider also a multilingual system where users of different nationalities and different character sets have accounts. The contents of the `/etc/passwd` file are going to be semi-incomprehensible to everybody. This kind of problem extends to many other areas. In the end, it will probably be necessary to retain ASCII functionality for system administration, or else each system will have to have a basic "nationality" that determines the character set used in system administration.

For the moment, these problems are immaterial, since only a single codeset has been specified thus far.

5. *Handling Mixed Codesets — Hardware Implications*

The X/OPEN NLS specification deliberately makes no attempt to address the device-handling problems that may result from the introduction of new, non-ASCII codesets.

No one wants to have to buy a completely new set of terminal and printer equipment in order to be able to use internationalised software. However, this is exactly what the introduction of NLS implies at those sites where the hardware does not support 8-bit character sets, or where 8-bit codesets are supported but not IS8859/1.

One can expect that it will be a long time (if ever) before the majority of terminals offer IS8859/1 codeset support. In the interim, users will want to make do with what equipment they have. As long as terminals and printers support alternate character sets some means can be found to force (maybe virtual) 8-bit character support.

If 7- or 8-bit devices are to be used, IS8859/1 can be used for internal purposes, and terminal and printer device drivers amended to use lookup tables that enable them to transmit translated character codes to output devices, along with appropriate escape sequences for swapping between codesets, and to decode and translate incoming escape sequences and characters from input devices. This problem has been addressed in the past[2] and the area is well understood.

6. *Limitations of the NLS Specification*

The specification for NLS was outlined for the first time in Issue 2 of the XPG in January 1987. Since the standard is very new, it is necessarily somewhat limited in scope. However, future editions of the XPG can be expected to extend the specification into new areas.

Only 22 UNIX commands are specified as having to guarantee processing 8-bit data correctly. It can be expected that this list will be extended in future to cover all the standard UNIX Section 1 commands.

Only one extended 8-bit character set is specified. This set does not provide support for languages that have mainly non-Roman alphabets — Greek, for example, or Hebrew. Further codesets will however be named at a later date.

The introduction of dictionary collation for sorting, etc., rather then machine collation means that the old regular expression syntax for character classes is no longer sufficient. A special syntax will need to be introduced, that uses generic `ctype(3C)` class identifiers. For example:

```
[A-Za-z0-9]
```

will need to be replaced by something like:

```
[(isalnum)]
```

The exact form of the syntax for character classes is in the process of being decided by a working committee.

The language announcement mechanism defined in X/OPEN NLS is not very flexible. It is possible to set a new language/culture/codeset combination with `nl_init(3C)`, but not possible to set only a single element of the combination. This means that "mix-and-match" environments are not possible. If a program wants to switch a single element of the local environment, the entire new environment must be loaded into memory. The ANSI XJ311 draft standard for the C programming language

proposes a more sensible solution to this issue (see the next section for more details on the differences between NLS and XJ311).

Problems of coping with mixed character sets and with hardware that does not support IS8859/1 are covered in Sections 4 and 5.

7. *NLS and the XJ311 Draft Standard*

The ANSI XJ311 Draft Standard for the C Programming Language specifies a number of "international" library routines.

The NLS and the XJ311 internationalisation specifications share the same general architecture. However, they do differ in some minor ways:

— NLS's `nl_init`(3C) function is replaced in XJ311 by the more flexible `setlocale`(3C), which allows programs to set and reset individual elements of the local customs data.

— XJ311 separates string collation away from string comparison — `strcmp`(3C) and `strncmp`(3C) remain unchanged, while an extra function, `strcoll`(3C), must be used to collate the strings before comparing them. This is a sensible separation of functionality.

— slightly different routines are used for handling date and time.

— NLS includes some enhanced I/O routines — `nl_printf`(3C), etc. — that allow parameters to be passed in variable orders to print routines, to support variations in word order between languages. These are extremely useful routines, surprisingly absent from XJ311.

Overall, there are no substantial or irreconcilable differences between the two standard specifications. It is to be expected that they will eventually converge — hopefully taking up the best features from both.

8. *16-Bit Character Sets*

At some stage in the future it is inevitable that NLS will have to take the question of the extremely large Far Eastern alphabets into consideration. Since they contain so many characters, these languages can only be represented with 16-bit (or larger) character sets.

16-bit implementations of UNIX already exist, especially in Japan, where a number of companies (including AT&T Pacific) have their own proprietary Kanji solutions. The problem here is suggested by the very word *proprietary*. Many different methods of implementing 16-bit character sets are used and it is difficult at present to predict which, if any, will eventually turn out to be a future standard. Meanwhile, it is a fact of life that there is little or no data compatibility between different vendors' systems, and that software must be rewritten, maybe substantially, to work on different vendors' machines.

Moreover, if UNIX needs a lot of work to make it 8-bit compatible, it needs an order of magnitude more work to make it support 16-bit character sets.

There are certain other problems that may or may not crop up depending on the way in which 16-bit character sets are implemented. These include, amongst others:

— in designing a 16-bit character set, there is a trade-off between efficiency of data storage and efficiency of run-time processing (i.e. either one or the other will deteriorate, in certain circumstances badly);

— byte redefinition may occur. For example, a "/" in the second byte of a character in a 16-bit filename might be taken as a directory delimiter;

— most implementations allow for a mixture of 8- and 16-bit characters, which may cause difficulties in character recognition (and thus especially in string handling), and

— due to the large size of the alphabets, dictionary collation may entail an excessive amount of processing (in fact, there are at least three possible different collation orders for Kanji).

The viability of UNIX in the Far East cannot really be assured until standard codesets are agreed for each country, such that reliable, portable software can be produced, and machines of all persuasions can talk to each other easily.

9. *The Future*

It is the intention of the X/OPEN group to continue to develop and extend the NLS specification. It is likely that NLS and the ANSI XJ311 standards will converge at some future point.

It is to be expected that eventually all the UNIX commands will have to be 8-bit clean. Other areas will be cleaned up to support 8-bit data, most notably and importantly regular expressions and the `curses` library.

We can also expect to see an internationally accepted set of translations of the UNIX utility error messages, hopefully administered by an international body such as the X/OPEN group. Currently, if any translations exist at all, they are proprietary translations, and so the same error message will appear differently on different systems. However terse and incomprehensible the original UNIX error messages may be, they do have the advantage of being (with some minor exceptions) identical on all UNIX systems — this is a great help to debugging shell scripts, and generally to feeling at home in the UNIX environment. It is obviously vitally important that standards are maintained in this area.

At some stage, NLS will be extended to include other 8-bit codesets, to support such languages as Greek, Turkish, Arabic and Hebrew. The latter two pose new problems, being right-to-left alphabets (e.g. how are left-to-right and right-to-left languages embedded within each other?). Eventually, 16-bit codesets will be introduced.

My personal view is that one day in the future, when storage and processing are *really* cheap, we will see the introduction of a single, literally global, 32-bit character set. This character set would be so large that it could contain every single character of every single alphabet in the world, with some left over for the extra-terrestrials when they arrive. Such a codeset would be capable of referencing other things beyond mere characters — graphical icons, colours, etc.. Gone would be the problems of having somehow to work out what character set someone is trying to communicate with.

"And ASCII shall lie down with Cyrillic..."

10. *Finally...*

Internationalisation has become a real market requirement now that UNIX is starting to become widely accepted and used outside the academic world. There has always been fierce debate as to whether UNIX is a user-unfriendly operating system. If that is true in any way, its solidly American bias must be the most user-unfriendly aspect of all. Despite its limitations, NLS does at least offer a *standard* solution to

the problems of internationalising software.

All the member companies of X/OPEN are committed to supporting NLS on their UNIX systems. According to Mike Lambert of ICL, speaking on behalf of the X/OPEN group at a recent UNIX seminar in London, the members of X/OPEN are committed to conformance with Issue 2 of the XPG by the last quarter of 1987.

Portability of software that utilises the NLS interface will thus be guaranteed across the entire range of X/OPEN companies' UNIX systems. In addition, it can be expected that other computer manufacturers will follow the X/OPEN lead and provide NLS on their systems.

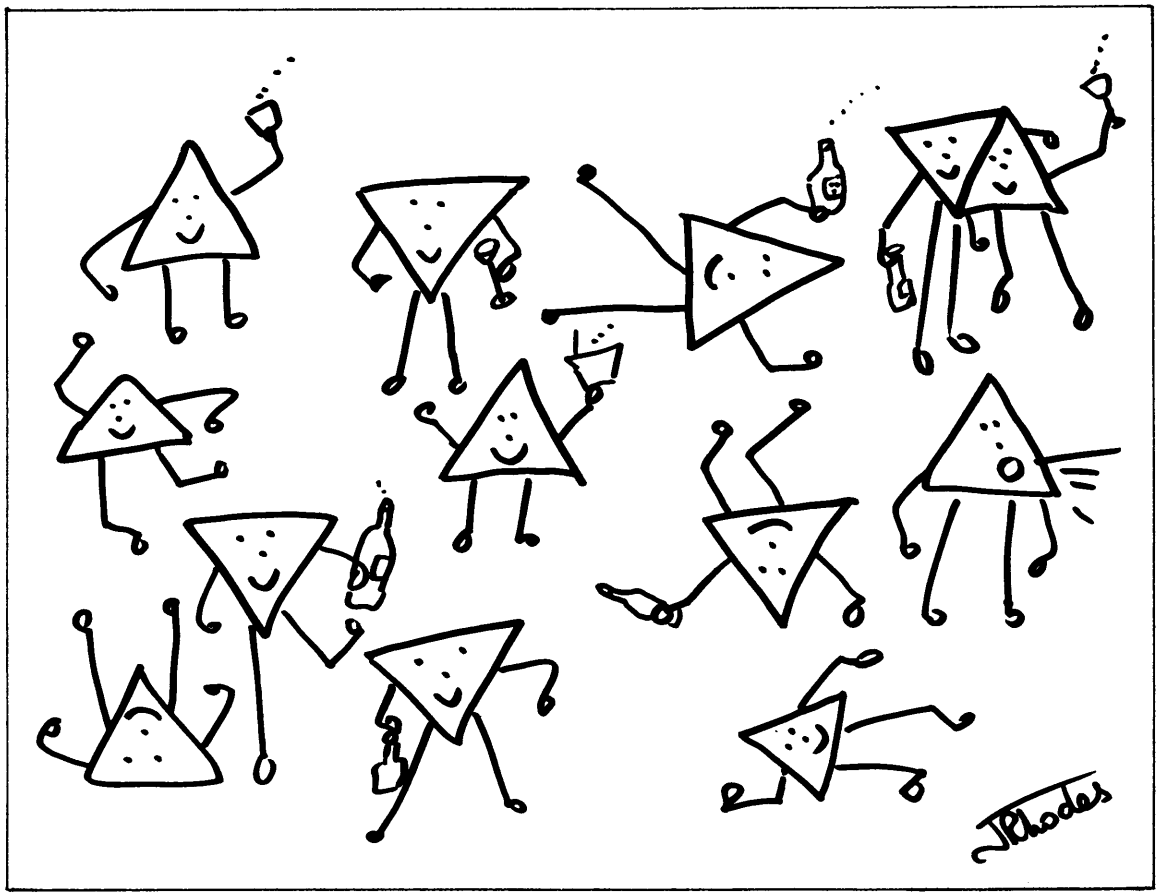
X/OPEN also guarantees a future upgrade path with backwards compatibility. This, in conjunction with the portability of NLS applications, ensures protection of software investment.

Although the present NLS specification is not perfect and is limited in many ways, it represents at least a pragmatic solution to the problem of internationally acceptable UNIX.

References

[1] *XVS Supplementary Definitions* X/OPEN Portability Guide, Volume 3, January 1987, Elsevier Science Publishers B.V.

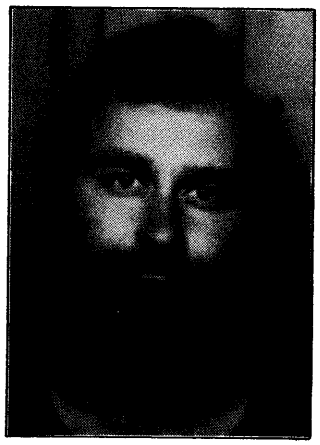
[2] **Conor Sexton** *European Languages in UNIX* Proceedings, EUUG Autumn 1985 Conference, pp. 195-210.



UNIX Grows Up

Grouse: Messages and Prompts in Programs

Alain D. D. Williams
...!mcvax!ukc!inset!phcomp!addw



Parliament Hill Computers

Alain Williams is an independent consultant specialising in UNIX and C. His interests are: finding what people's problems are, and building good tools to solve them once and for all; doing the job properly as it takes less time; drinking cider; and jumping out of small boats into a cold sea.

He is the editor of the EUUG newsletter.

Editor's Note:
This is a reprint of a paper which was given at the UKUUG meeting, 15 December 1986.

Grouse can be used in programs to replace printf() statements, the text for the messages coming from files. Printf() style argument substitution is available, arguments can appear in any order.

This paper discusses:

— *What grouse is like to use.*

— *Implementation.*

— *The advantages of choosing compiled text files.*

— *The way in which grouse is used by higher level functions to permit automatic interpretation of errno in error messages.*

— *A greatly simplified prompting procedure allowing help to be taken with little action by the applications programmer.*

— *Language independent option prompting trivially supplied.*

— *The way this is tied into a help package.*

— *How grouse can easily crash a program and why this is, in practice, not a problem.*

— *Portability of code and message files.*

Version 1.4
Updated 12/9/86

I like work: it fascinates me. I can sit and look at it for hours.
Jerome K. Jerome

Grouse: (noun) grumble (slang) (Oxford English Dictionary).

1. *Introduction*

What follows is basically an exercise in laziness, one in which I was prepared to invest a large amount of effort. There are two major topics in this paper: `grouse` a text from files subroutine, and a set of subroutines built up on top of `grouse`.

1.1 Ancient History

The initial impetus came from a set of routines which I first wrote in 1979. These were to help me debug BCPL programs. I put assert type traps in them which printed a rude message, and quit. Often needing different amounts of information I soon got bored with recompiling, something which took a long time — even with an illegally raised priority.

Editing text files was fast. So I put all the messages into a file (along with flags requesting things like dumps, and stack traces) and wrote a routine to find and print numbered messages. Because this was always used in a situation where programs complained about errors I called the routine `grouse`.

The next ingredient is numbering arguments. Thus referenced I could substitute and format them using `writef`[1]. This is easy to do in BCPL, which implements a word based machine architecture: I just indexed on the address of the first argument.

I then became a UNIX® acolyte and implemented `grouse` on a PDP11. The way the C compiler worked allowed the old indexing trick, but different objects had different lengths. This caused problems. I wrote a compiler to fudge the issue, it massaged the numbers. A compiled file also offered greater efficiency of message location and meant that the run time code could do away with much format checking — code size ever a problem on a 64K machine.

1.2 The Present Day

As a simple message system `grouse` is nothing unique. Its great usefulness has come from a set of subroutines which all use `grouse` as a base. The most interesting are those concerned with prompts in interactive programs. The programmer is presented with a simple, clean, standard interface; he need not care if the actual form of the prompt is changed to suit the flavour of the month.

`Grouse` and its family are in the throes of a third birth. This opportunity has allowed me to try and avoid past problems (and introduce new bugs). It is this version that is described here.

2. *Overview*

This section should give you an impression of what using `grouse` is like, and the effort involved in using it.

1. The BCPL equivalent of `printf`.

2.1 *The Bad Old Days*

This is the era in which most programmers still live. Two examples follow: they are fragments of code typical of what I have seen many people generate — though they usually don't have as many comments.

```
fprintf(stderr, "%s: Cannot open '%s' as %s\n", progname, file_name,
        sys_errlist[errno]);
```

Figure 1. An error reported

```
name = "master";
for(;;) {
        position(22, 0);
        line_clear();
        position(23, 0);
        line_clear();
        position(22, 0);
        printf("Oh %s: shall I rename %s as %s ?\nReply (Y)es or (N)o : ",
                name, old, new);
        switch(getchar()) {
        case 'y':       /* He agrees with me ! */
        case 'Y':
                .
                .
                break;
        default:        /* Idiot can't read the options */
                position(21, 0);
                line_clear();
                position(21, 0);
                printf("Please reply with one of the options below");
                name = "idiot";
                continue;
        }
        break;
}
```

Figure 2. A Question Asked

Notes on the examples:

— The message text is embedded in the program.

— In Example 2, the programmer needs to cope with the error case himself. This results in extra, unwanted, code. A `for` loop is needed with resultant extra indenting, control structures, and so complexity.

— An explicit test needs to be made for both upper and lower case.

— There needs to be screen positioning code, etc.

2.2 *Modern Times*

With `grouse` the two examples may be coded like this:

```
                /* Report the open failure */
egrouse(stderr, GR_OPEN, (char*)errno, progname, file_name);


                /* What does he want to do ? */
switch(gaskq(GR_PROMPT, (char*)0, name, old, new)) {
case 1:         /* He agrees with me ! */
        .
        .
        break;
}
```

Figure 3. Grouse

Notes:

— There is less code to type.

— The message text is not obvious.

— The reply has been checked, there is no need for an error condition.

— Both prompt and reply are language independent.

— Help comes free, the programmer does not need to do anything to get it.

— The second argument is always `errno`, even if it is not wanted. It must be cast as shown.

Unfortunately the immediacy of having text in the code is lost. You have to look in a message file to inspect or create that text. The messages are referenced symbolically by `#defines` — which are probably contained in a third file. This requires a disciplined approach and a little documentation.

`Grouse` can also be used to obtain constant text which may be expected to change from one situation to another; e.g. the names of days of the week, customer name, file/directory names.

3. *Message Files and Message access*

A message file exists in two forms: source and compiled.

There are two compiled files that `grouse` may use at any one time. The idea is that one contains text that may be common between a group of related programs. `Grouse` decides from which file to obtain the message on the basis of the message number: numbers in the range 0 to 19999 are read from the first file, numbers 20000 and up from the second.

The way that I have used this feature is to put in the first file text used in many places, e.g.: strings representing the values of `errno`, prompts for help routines (being library routines common to a suite of programs), a language collating sequence.

There may be large gaps in the message numbering sequence. This is very important in providing a stable environment to existing code, yet allowing new messages to be added to a common part. In previous versions a break in the sequence meant a corresponding waste of disk space.

3.1 How to Find a Compiled Message File

This is a big key to (human) language independence. The name of the file opened is built up of: an application supplied `grouse` file name; a constant part; a suffix, and two environment variables. These variables are `LANGUAGE` and `MSG_LEVEL`. Standard uses are: the first names the user's mother tongue, and the second is a measure of his incompetence.

In this way an individual's environment can be set so that he uses the machine (reading, and replying) in his native language. Similarly it is trivial to arrange for a beginner to see prompts that would be unacceptably long to an expert. So two people running the same application, on the same computer, at the same time could see something very different.

3.2 The Source Message File

This is a plain text file created with a suitable editor. The messages are delimited on a line basis. They contain the text that is to be retrieved by `grouse`; this may be several lines long. The backslash \ character is used in a similar, but extended, way to the C language.

As with `printf`, arguments are introduced with a `%`. However, the character immediately following indicates the position of the argument as passed to the top level function. If any arguments are unused, what they are must be indicated in a special way.

There are several different message formats. This corresponds to the different higher level function that will end up using it. The type will be checked by the message compiler.

What does it look like?

```
%1s: Cannot open '%2s' as %0s
```

Figure 4. Grouse Text for the First Example

```
Oh master: shall I rename %1s as %2s ?
Reply (Y)es (N)o (H)elp : ^ |y|n|h?rename_help
```

Figure 5. Grouse Text for the Second Example

Notes:

— The arguments are indexed by the character after the `%`.

— The arguments can occur in any order.

— You don't have to use the first argument.

— Message length, in lines, is not important to the application programmer.

— In choice prompts the reply is encoded in text after the prompt.

— Help is obtainable by entering `H`. The file containing the help text is also encoded as part of the reply text.

3.3 The Compiled Message File

This is what the `grouse` subroutines see. The main advantages of a compiled message file remain as they were originally:

— The ability to work on a non-word-based architecture. The message numbers are changed to make the old, simple indexing scheme work. With some awkward

machines it is necessary to also include extra information on length and positioning.

— The message file contains information that allows `grouse` to calculate the file offset of the desired message. This is much more efficient than sequentially searching for it.

— The compiler has syntax-checked the messages, and made them easier to parse for use.

The compiler offers facilities such as argument type cross check (with the corresponding message in another file) and symbolic referencing of one message to another.

There is also a certain amount of control information, including version numbers and the compilation date.

4. *Higher Level Functions*

This is where the fun starts.

4.1 Egrouse — Errno Interpretation

You have already seen this in the new coding method for the first example. `Egrouse` is functionally the same as `grouse` except that it takes its second argument to be a value of `errno`. It calls a function to return a string into a secondary buffer, which provides an interpretation of the number. It then overwrites that argument with the address of the secondary buffer and calls `grouse`.

It is because of this overwriting that, as you will have noticed, `errno` must be cast to `char*`. Many of the high level functions use `egrouse` and so need a cast value of `errno` passed to them. This is initially confusing, but soon becomes natural.

If the situation has no use for a value of `errno` one must still be provided. In this case zero may be passed. The empty string will then be assumed, i.e. no file access will occur.

4.1.1 Errno Extended

Not being restricted to the list of messages that I found in `sys_errlist` came as a boon. I was able to easily add my own definitions of values for `errno`.

Up until now if something went wrong in a library routine it had to report the problem itself. This might be difficult if it was something of general use — what method should it use, should it complain at all?

All that was needed was for the routine to set something meaningful into `errno` and return failure, this being transmitted up a calling sequence. At some convenient level, and if appropriate, the problem could be reported with `egrouse`. It made no real difference whether the error status was one of my own invention, or one from the vanilla list.

4.2 Gaskq — Prompts

A common event in an interactive program is prompting of the user for information. This can occur in several different ways, the most common of which is a prompt for a choice of action. This is what was happening in the second example. `Gaskq` `egrouses` the message into a buffer and calls `askq`. The latter puts the message on the terminal and looks at the reply.

A list of acceptable replies is encoded at the end of the message. If a match is found the default action is to return the position in the list of the matching reply.

Thus, in the example, `y` returns one and `n` returns two. This action may be changed by following the reply by an action string. The one illustrated directs `askq` to call the help function, and gives the name of the help file to use.

Other actions include: reprompt with another prompt and return a number other than that deduced from the positional rule. It is possible to specify a default.

Related functions prompt for a string or simply display a message.

4.3 Help

One of the `askq` built-in actions is the provision of help. This is obtained through a standard interface from `gaskq`. The default help routine allows the user to peruse or dip into the help text as he wishes, moving back and forth in the text, following links into other files for explanation of common parts.

The routines prompt the user to direct his perusal using `gaskq`. Help on how to use help is trivially provided.

A "mini help", i.e. a slightly longer, more explicit, prompt can be obtained by switching to another message number to use as prompt. This will be displayed, again using the same arguments as the original.

The calling program knows nothing of all of this as it is done by `askq` directed from the message file.

4.4 List Building

There are occasionally lists of words that a program may need to get going, for instance the names of the days of the week, or column headings. These can be obtained from `grouse`.

The action in each case is similar, i.e. read the text, allocate storage, and split it up into different elements of a string array. A routine is available to do this, and it checks more than one might normally do on a once off basis: are the number of elements between certain limits, is any element too long?

5. Programming Considerations

As you have seen `grouse` may be used in a similar way to `fprintf()`. To be useful in the higher level functions, the text needs to be put into a character buffer — as with `sprintf()`. The trick employed is that if `grouse` sees that the message number is negative it considers its first argument to be a `char*` rather than a `FILE*`.

The programmer has to specify the names of the two `grouse` files. This is managed by setting a name into a global structure. There is no need to explicitly open the files.

The defined arguments to the open routine are a name and a suffix. This routine is thus available for finding other sorts of file in a similar way, for example, help files. If screen layout files are used, this area of the MMI can also vary in the same way as the messages and prompts.

5.1 Ranargs

This is a facility which enables the arguments to a function to be accessed in a random order. It is designed to have a feel similar to `varargs`.

Define an argument list as being the arguments that a function has, possibly omitting a specified number at the start. There are two things that `ranargs` lets you do with a list of arguments: access a specific argument, and pass the list to

another function. The latter ability is vital in constructing the higher level functions.

This is all simple enough on machines where the arguments are placed, one after another, on the stack, and there is no greater alignment needed than that of an integer. If these conditions are not met, the `ranargs` macros become complicated, possibly invoking functions. To get some idea of the problems involved consider why, on a machine where doubles indeed have a greater alignment than an integer, the compiler may have to leave a gap on the stack before it.

The lack of exact control over what the compiler does is one of the consequences of using a high level language. The problems I have had are similar to those who try to do single precision floating point in C, and I know of several device drivers that cannot be optimised.

5.2 Errors

As with `printf()` `grouse` returns -1 if it can't do what it was asked to. Frequently programmers don't bother to test return values because "It always works", and because it is a chore. The problem becomes even more difficult if an error is detected by a higher level function; what should it do?

Firstly, the external result is always sensible. Even if the function's return value is ignored there should be no bad effects, e.g. `grouse` never leaves an unterminated string. Next, on finding trouble all the routines in the package call an error function, suitably indicating the problem. The library default of this function does nothing, but it can easily be replaced by the application programmer.

5.3 Robustness

The code in `grouse`, and the higher level functions, has proved solid. The problems that are likely to arise are in their use; probably not in the coding itself but through the message file — particularly if this gets changed. The three areas with greatest potential for `grouse` crashing a program are:

1. Inappropriate use of an argument.
   For example trying to output an integer as a string. In practice this is rare as the programmer will normally create the message file when he writes the code. If the messages are changed by someone else, the compiler cross-check action can be used to pick up any mistakes.

2. Overwriting the end of a string buffer.
   If text is going to a character buffer `grouse` does not check that it does not overflow the buffer. In four years of having a primary buffer of 550 bytes, and a secondary one of 150, I have not experienced any problem because of this. The choice of buffer size is important and needs to be carefully considered; those changing the message file should be aware of the problem.

3. A message not being found.
   This could be because the message file is not accessible, or a message is missing. This occasionally happens (often in `gaskq`); the usual result is that at that point the program always takes a certain branch — occasionally leading to an infinite loop. This situation is now easier to trap with the introduction of the `grouse` error function.

   If the file exists, the usual cause is running a new program with an old file. The version number feature is designed to avoid this.

5.4 Malleability

Much recent effort has been made in the area of ease of programmer customisation: how he can best make it do what he wants. This has been done without making simple or standard use any more difficult. The programmer has much control, but only if he wants it.

Some standard routines can easily be replaced by the programmer. One version of the `grouse` open function that I have searches down a path taken from the environment in a similar way to the shell when finding commands.

`Askq` (and hence `gaskq`) is independent of the method of communication with the terminal. Thus by replacing the low level display routine it is easy to change the user's view of the program from dumb terminal, to a cursor addressed prompt line scheme, to a mouse-driven pop-down menu approach; all involving no change to the main body of the program. It may be expected that the message file may have to change to match the different styles, though I am working on a way of transparently incorporating alternatives into the message file.

The `grouse` package is designed to cope with 16 bit characters. There is no hard-wired dependence on special characters or character size; these may exist in utility functions called by the package. There is still much work to do in this area: I have still to find a 16 bit terminal on which to test this.

5.5 Portability

There are two aspects here: the code and the message files.

With the invention of `ranargs` the code has proved to be portable. Other than this there is little in the code that is contentious — except to `lint` which is vastly unhappy about some of the strange type casts.

The message files are only portable in source form. This is because of the massaging needed on message number.

6. How does it Work in Practice?

A version of `grouse` has been available for four years. During this time it has been used on two major projects, both involving several large programs, and a total of some 1500 messages. The latest version is too young to have been used in anger, but was designed to alleviate problems or restrictions in earlier attempts.

6.1 Programming

The main problem has been one of initial learning and acceptance. People get set in their ways, and I found that, even with some encouragement, some are reluctant to use new tools. There is a short learning phase (`grouse` has always been well documented), and a certain discipline is required to ensure that the symbolic numbering does not get out of step with what is in the file. Once this has been done the attitude changes to "this is the easier way".

If a model of interaction is chosen that is not available as standard, then some new low level display routines need to be written. This can be a barrier to those with the "I'm only interested in today's problem" attitude — especially if they know that someone else is to maintain and possibly modify the code for another language.

6.2 Speed

I have not done much quantitative analysis. However, is it pleasing to report that an interactive `grouse`d program does not appear to be significantly slower than a hard-wired equivalent. This is even in a situation where a composite message may

be produced from several fragments.

6.3 *Altering Messages*

Documentation is the key. If this is not complete and explicit then references need to be made to the code: having a quick peep is error prone — even for an experienced programmer. This applies if the text is being translated or the style is being changed. Rewriting a `grouse` file is not a quick and easy task. Message files tend to be large, and the translator needs some understanding of `printf`. I have found that, with a little tuition, even managers can do this.

7. *Conclusions*

— It has been worth while. The increase in productivity through using the tools discussed here is great. More work is needed to take them further, both in the programmers' tool kit and to simplify the task of the application customiser.

— C has been criticised for being too low level a language for applications programming — I quite agree. However lack of precision can cause problems, as evidenced by `ranargs`. What to do is not obvious. C has served us well and doubtless will for some time yet.

— There are many things which I find boring: writing similar code several times is one of them. Subroutines were discovered years ago. I am still surprised by how often people who ought to know better will still "lift and hack" code rather than "doing it once and doing it well".

Another Proposal for a News Scheme

John Collins
jmc@xisl.co.uk

Xi Software Limited

1. *The Problem*

There are at present two options regarding the news:

1. Don't get it at all.
2. Get all of it.

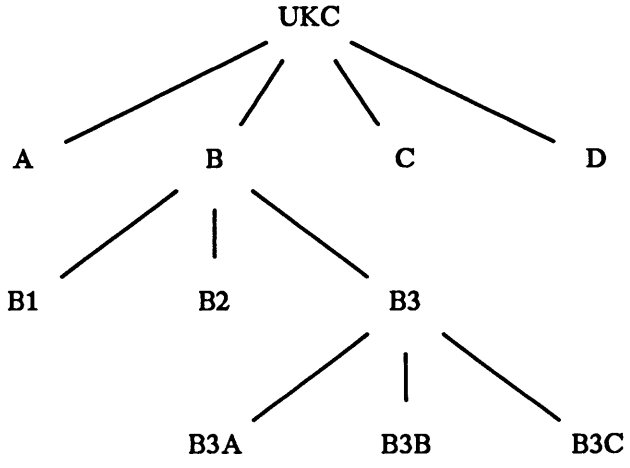
Assuming that the reader wishes to avoid option 1, option 2 may be undesirable for any or all of the following reasons.

1. There is so much of it he/she doesn't have the disk space.
2. It takes so long to transmit that the phone is constantly engaged.
3. The phone bills involved are prohibitive.
4. Having got it, about 5% is interesting and the remainder is about someone selling a second-hand goldfish bowl on the other side of the world (usually an American with an imperfect understanding of the relationship "usa != world").
5. People at various sites spend hours every morning wading through all these news items trying to find interesting ones instead of doing useful work.

The following is my suggestion for a way to compromise between the two extreme options with regard to the news.

2. *The Proposed Solution*

For the sake of argument, let us suppose that a section of the network is as follows:



At present, sites A to D obtain news from UKC, and site B sends a copy to each of sites B1 to B3, site B3 sends further copies to B3A to B3C. This represents a great

deal of duplication — especially if only a few items are of interest at each site.

The proposal is that site B continues to receive the news from UKC as at present (note: no work for UKC!), but instead of passing it on, passes a "headline file" to each of the sites on the line. This file is a text file, derived from the newsgroup name, the article id and the subject line, possibly of the form:

```
mod.sources

xyzabc.1234     Public Domain ADA compiler Part 01/87
xyzabc.1235     Public Domain ADA compiler Part 02/87
    ......

misc.rumors

foobar.1234     AT&T to give free source licence with every 3B
blech.1383      IBM to pull out of computers
frozb.3934      Re: IBM to pull out of computers
    .....
```

Users on each site then put up this file and request any articles of interest, using a full screen program. The requests are collected at each site one down from the "headline sender" site. Thus each of sites B1 to B3 collects requests relating to the "headlines" sent by site B.

At the sites B1 to B3 the requests are "weighted" using some or all of the following factors:

1. Number of requests for each item.
2. Age of item (so that old news decays in importance)
3. Cost of transmission to requesting site.
4. News group importance.
5. Item size in bytes.

For each site B1 to B3 there is a "request threshold"; if the total weight of requests exceeds this, then the item is obtained from site B.

Likewise for each site B3A to B3C there is maintained at site B3 a threshold for passing the item on.

This can all be extended downwards, say from one of sites B3A to B3C, and in that way, subject only to a day or so's delay in getting the news, some control is obtained in the volume of the news.

UKUUG 1987 Summer Technical Meeting

Sunil K Das
sunil@ucl-cs.ac.uk

UKUUG Chairman

Date:	PM Tuesday 7th - AM Wednesday 8th July 1987
Programme Chairman:	Sunil K Das
Host:	Lindsay Marshall
Venue:	Newcastle University
International Speaker:	Michael Lesk

1. Provisional Programme

Tuesday 7th July, 1987

1100			Tour of the Computer Laboratory (optional)
1230			Registration Begins
1300			BUFFET LUNCH Begins (optional)
1400			Meeting Begins
1400	-	1420	"Implementing a Turnkey Package under UNIX" Alain Williams (Parliament Hill Computers)
1420	-	1520	"Route Finding in Street Maps" and "Keywords vs. Menus in a Library Catalog Interface" Michael Lesk (Bellcore)
1520			TEA
1600	-	1630	"High Performance UNIX Multiprocessor Systems" Peter Lee (Newcastle University)
1630	-	1700	"Integrating the Apple Macintosh in a UNIX Environment" Nick Nei (Glasgow University)
1700	-	1720	"Opening Windows on UNIX" Paul Davison et al (Newcastle University)
1730			Adjourn to the bar DINNER — waitress service inclusive of wine Adjourn to the bar

Wednesday 8th July, 1987

0800	BREAKFAST
0930	Meeting Reconvenes

0930	-	0950	"Recoverable Object Management in Arjuna (using C++)" Graeme Dixon (Newcastle University)
0950	-	1050	"Computer Technology Forecast" Michael Lesk (Bellcore)
1050			COFFEE
1130	-	1200	"Experiences with MINIX and Networking" Jim Lyons (Newcastle University)
1200	-	1220	"UKUUG Business Meeting" Sunil K Das (City University, London) • Election of Officers • Presentation of Accounts • News from EUUG
1215	-	1245	"Improved Models of Natural Language for Consultative Computing" Eric Foxley and G M Gwei (Nottingham University)
1245			Meeting Ends
1300			BUFFET LUNCH (optional)
1400			Tour of the Computer Laboratory (optional)

Michael Lesk is well known for etc...(see vol. 2 of the 7th edition manual and the 1978 UNIX BSTJ).

He will speak for 1 hour on Tuesday on some of his recent research:

- "Route Finding in Street Maps"
  This talk explains a computer system for giving people driving directions, using machine-readable maps. It also compares how computers find routes with how people find routes.
- "Keywords vs Menus in a Library Catalog Interface"
  This talk compares two interfaces for an online public access catalogue. It's a software psychology experiment with real data.

On Wednesday, Michael will speak for another hour:

- "Computer Technology Forecast"
  Bellcore keep asking him to predict the future.

Glasgow Local UNIX Group

Zdravko Podolski
zp@glasgow.cs.ac.uk
zp@glasgow.uucp
...mcvax!ukc!glasgow!zp

Computing Science Department,
University of Glasgow,
UK

The group consists of about half a dozen companies, several departments of the two Universities and a few other Government and local council funded bodies such as Paisley College of Technology. We have been meeting regularly for about six months now. We meet every 2nd Tuesday of the month at the premises of one of the members, when the agenda consists of a short presentation of what someone is doing, followed by a discussion of which pub we should proceed to (hence the name). We welcome all interested parties to come and participate. There is no charge and we do not see the need for introducing one. There is no formal committee structure, we operate in a spirit of cooperative anarchy. A tape of free software and the compilation of hardware and software so that folk can get access to things for evaluation are being put together. We help and encourage folk to get networked to UKnet and/or JANET for news and mail feeds.

Some of the talks we have had were:

Jim Reid Strathclyde University	Electronic Mail and addressing conventions
Mick Hughes Turing Institute	Typesetting using TeX
Nick Nei Glasgow University	Networking Macintoshes and UNIX machines
Jim Reid (again)	NFS
Allan Birse Strathclyde Electrical Engineering	Cryptography and UNIX

Anyone interested in coming to our meetings is invited to contact:
Hossein Baniamer, ABACUS, Strath. Uni. 041-552-4400 extn 3024 or
Jim Reid at the Computer Science Department, University of Strathclyde,
041 552 4400 extn 3319 or
Zdravko Podolski, Computing Science Department, University of Glasgow,
041 339 8855 ext 4469.

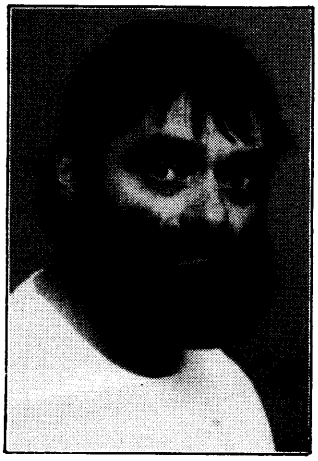
Email addresses:

hossein@uk.ac.strath.abacu (no UUCP! and no "s" at the end either)
jim@uk.ac.strath.cs, jim@strath-cs.uucp, ...!ukc!strath-cs!jim
zp@uk.ac.glasgow.cs, zp@glasgow.uucp, ...!ukc!glasgow!zp

EUUG

Teus Hagen
EUUG Chair
teus@oce-rd1.nl

EUUG executive board



Teus Hagen was born in 1945. He graduated from the University of Amsterdam. His computer science knowledge was gained mostly at the Free University of Amsterdam.

He has been involved in UNIX since 1975, at the Mathematical Center (currently CWI) in Amsterdam (the second UNIX site in Europe?).

Since 1985, he has worked in UNIX companies, and is currently at Oce Holland (Office Automation).

Teus has been involved with the EUUG since 1977, and has been the EUUG chairman since 1985. He was responsible for starting Eunet at Paris in 1982, and arranging the current structure of the EUUG in 1983.

1. Academic, that is a long time ago

Some old hackers say the group started in Edinburgh, others say it was in Canterbury, continental hackers believe it was in Amsterdam, but nevertheless all are right. It was about ten years ago that some enterprising people started to complain about UNIX. They exchanged software on huge disk cartridges in order to adjust their feelings for the next time. From about 15 academic students and professors from a few countries in Europe, the EUUG went in ten years to 2500 members (mainly from institutions and industry) from almost every country in Europe. In *Dublin* the EUUG will celebrate its tenth anniversary. That event should not be missed!

Apart from the countries behind the iron curtain, and Portugal, a national UNIX group can be found in every country, all bundling their common efforts under the umbrella of the EUUG. Each national group has appointed her representative to the *governing board*. The daily bother is left to eight persons in the *executive board*. They meet about every two months in order to keep the EUUG going. The secretarial assistance comes from the people at Owles Hall in England. Those people really make a good job of all the peculiarities arising from all those languages and national specialities.

With the big growth and the success of the organisations and the EUUG one can expect some growing pains. On the national level, there is plenty work in combating the national problems. This means less effort to be expended on the European level. Also the enormous amount of work which accompanies the EUUG work is very hard to see. Conferences need a lead time of at least one year. To get plans adjusted and agreed takes a very long time on the European level. The effect of all that work is seen too late. Still, the enormous encouragement from the national level is a big motivation for the executive board to continue.

COUNTRY	GROUP	MEMBERS	COUNTRY	GROUP	MEMBERS
Denmark	DKUUG	164	Britain	UKUUG	304
Finland	FUUG	131	Ireland	IUUG	43
Sweden	EUUG-S	195	Swiss	UNIGS	28
France	AFUU	362	Germany	GUUG	392
Italy	I2U	291	Austria	AUUG	20
Holland	NLUUG	190	Belgium	BUUG	31
Iceland	ICEUUG	10	Norway	NUUG	68

EUUG Membership January 1987

Normally the time spent by board members on EUUG tasks is too long (usually about 20% of their working hours). One can expect that more full time paid personnel will be needed in the future to limit their EUUG working hours.

Clearly the conferences, with their technical contents, industry presentations, and tutorials, are the first things from the EUUG one sees. The EUUG newsletter is felt not to be distributed frequently enough. More effort is being put into meeting that problem. More EUUG publications similar to UNIGRAM weekly, a glossy journal (probably UNIX Review), a UNIX technical journal, a catalogue and a European UNIX diary, can be expected.

Unexpectedly, the EUUG software distributions are meeting the needs of the members. There are thoughts of making that software available on other media as well as magnetic tapes. Also there are discussions going on about making them available on EUnet.

EUnet is being registered as a trademark throughout Europe. The network is recognised as one of the largest in Europe. Gateways exist to other networks such as CSNET and DFN. For every country a domain is being registered (was Holland with the domain *nl* the first?). The backbones are very well interconnected. Sometimes I'm a bit angry that my electronic mail takes more than an hour to get answered by someone in France. Sometimes I get worried that the people of a particular company in the US have gone home before they have had a chance to read my mail. Due to the efforts of the people involved directly with EUnet, email has less failures than one could imagine some years ago. EUnet is very successful.

Sometimes it is necessary not to put some of the work in the direct sunlight. That certainly is true for the regular meetings with the organisations in the front lines of the UNIX fields: for instance UNIX Europe, AT&T International, but also X/OPEN. Especially, perhaps, X/OPEN needs some more input from the user group. More manpower is needed to support that effort. Think about the enormous experience Europe has with international peculiarities (character sets, languages, etc.).

The EUUG has recovered from her past financial problems. The financial situation is sound now, so we can direct our attention more now on those problems and tasks which are common throughout Europe. For just that reason, there is an EUUG. The enormous stimulation, the contributions, from the national groups make this UUG organisation a success, which should not be missed in any country.

Progress of ANSI/ISO C Standardisation

Cornelia Boldyreff
...!mcvax!ukc!reading!uoseev!corn

Department of Electronic and Electrical Engineering
University of Surrey
Guildford GU2 5XH

1. *Brief Historical Background*

There exists a high degree of homogeneity between various implementations of C for a variety of reasons:

- the common origins of C compilers;
- its link with the UNIX system (it has been remarked that successfully compiling the UNIX system is quite a rigorous test for a C compiler);
- C is used as a vehicle to achieve portability in general which militates against adding non-standard extensions;
- Dennis Ritchie and Brian Kernighan's clear exposition of the C language.

The latters' book, "The C Programming Language", has become an informal standard for the language; it is not uncommon for suppliers of non-UNIX C compilers to assert that they support Kernighan and Ritchie. This book was published in 1978 and as a standard is becoming somewhat outdated.

It formed the base document for the ANSI C standard committee's development of a standard for C. They also drew on work by the US commercial UNIX users group, /usr/group; particularly for the definition of the C library. The ANSI effort has received support from AT&T as well as major C compiler developers, suppliers and users including UK companies: ICL, The Instruction Set and Edinburgh Portable Compilers. In December 1985, ANSI proposed a New Work Item on C to the International Standards Organisation based on their work.

2. *Introduction to Work of the BSI C Panel*

The BSI Technical Committee on Programming Languages, IST/5, had been monitoring the progress of C standardisation efforts prior to the proposal by ANSI of an ISO New Work Item on the programming language C. Following approval of the C NWI in April 1986, an ISO Working Group on C was formed; and the formation of the BSI C Panel was set in motion. The role of the C Panel is to provide a UK focus for contributing to the progress of an ISO Standard for C. This panel met for the first time in the summer of 1986. The first meeting of the ISO Working Group was in September 1986; countries represented were the USA, Canada and the UK. The ISO standard work is progressing in parallel with that of the ANSI X3J11 Committee. It is very much a collaborative effort as the draft proposed ANSI C standard is the basis for the ISO standard rather in the way that the BSI Pascal standard was the basis of ISO Pascal.

The BSI C Panel

— is a representative group of UK experts including commercial, industrial and academic users of C as well as suppliers and developers of C compilers;

— meets informally and has no official BSI status; all members of the panel act in a voluntary capacity usually supported by their employers;

— reports regularly to the BSI's Technical Committee on Programming Languages, IST/5, through its Convenor who is a member of IST/5 on its activities and the progress of the C standard;

— advises IST/5 on issues concerning C and ISO ballots relating to C;

— monitors progress of and contributes to the ANSI work on C;

— contributes to the progress of the ISO C Standard through participation in the ISO Working Group on C as individual experts with "awareness of UK reactions".

— collaborates with other UK BSI committees concerned with C related standards work; for example, graphics standards with C Bindings, and the proposed POSIX standard.

The C Panel meets quarterly preceding ANSI (ISO) meetings and BSI IST/5 meetings. Panel meetings are usually attended by a dozen or so members — the panel officially has 16 members. New members are always welcome. Membership is considered to lapse if a member does not attend for three consecutive meetings of the panel. (Interested parties could contact the author who is convenor and chairman of the C Panel.)

3. *C Standard Open Meeting*

The British Standards Institution's C Panel organised a one-day "Open Meeting" on the proposed C standard to coincide with the BSI's publication of a Draft for Public Comment on the programming language C. The meeting was held on the 9th February 1987 at City University, London.

The keynote speaker at the meeting was Dr P. J. Plauger, President of Whitesmiths Ltd. Bill Plauger is a prominent member of the ANSI committee, X3J11, which drafted the proposed C standard, acting as secretary to X3J11; and chairman of the C library sub-committee. In his opening lecture, he gave delegates an overview of the current draft C standard concentrating on major decisions reached by X3J11 and issues which had taken up the most "air time" in committee meetings over the past three years. He enumerated the tenets of the philosophy which has guided X3J11 in their efforts to standardise C as follows:

— Codify existing practice.

— Existing code is important.

— Portability needs a "fighting chance".

— Non-portable code is OK, too.

— Quiet changes (to C) are bad.

— The standard is a treaty between implementor and programmer.

— The "Spirit of C" is important.

He discussed three major decisions made by X3J11 regarding characteristics of the C machine, C programs' conformance to the standard, and implementations of C. A "C machine" has 8 bit or larger bytes; "no holes" in objects (i.e. except for bit fields, objects are contiguous sequences of bytes); performs weighted binary arithmetic; and has an arbitrary character set. A C program is either strictly conforming to the standard i.e. fully portable; conforming; undefined, or erronious. An implementation of C may be either hosted or freestanding.

The standard has endeavoured not to radically change the C language; major issues which have concerned the committee identified by Plauger were:

- Conformance issues;
- Widening rules;
- Preprocessor issues;
- Library issues.

In conclusion, Plauger explained the rationale behind the introduction of function prototypes to the C language by elaborating the committee's own version of the US Supreme Court's Miranda Ruling. He also listed extensions considered by the committee which failed to gain approval showing that there was scope for framers of the C standards to come in the 1990s.

Cornelia Boldyreff, the Convenor and Chairman of the BSI C Panel, spoke briefly introducing the work of the BSI language panel concerned with C standardisation. The BSI C Panel was formed in the summer of 1986. Its role is largely advisory; it advises the BSI's Technical Committee on Programming Languages on issues concerning C and ISO ballots relating to C. It monitors progress of and contributes to the ANSI work on C; and panel members contribute to the progress of the ISO C Standard through participation in the ISO Working Group on C as individual experts with "awareness of UK reactions".

The morning session was concluded by John Souter of the BSI's Certification and Assessment Service addressing the issue of testing conformance to standards by language processors. He outlined the work of the BSI evaluating potential candidates for a C Validation Test Suite. According to Souter, the USA validation service is planning to follow the British lead in establishing a test service for C language processors.

The afternoon session was given over to discussing the three main aspects of the standard, dealing with the C language, the C library, and the C preprocessor. Mike Banahan of The Instruction Set addressed the C language and the C preprocessor in two lectures; and Bill Plauger spoke again in greater detail on the C library.

Banahan reiterated that it was not the intention of the committee to radically change the C language; he reassured the meeting that much of Ritchie's original description of C could still be found in the text of the draft standard. His lecture concentrated on illustrating key points where the language has changed.

In his introductory remarks on the work of the library sub-committee, Plauger mused given his involvement in the development of the Whitesmith's C library, some must have seen his selection as chairman of this group as comparable to putting "a fox in charge of the hen house". Of particular interest to the international C user community were the ways outlined by Plauger in which the committee had addressed the issue of "Internationalisation" by inclusion in the library of a runtime selectable locale. The library defined in the draft standard has had ASCII dependencies removed; is more complete; and covers domain and range errors in mathematical functions. Areas of the library identified by Plauger as still needing attention included functions to restore calling environment: `setjmp/longjmp`; signal handling: `signal/raise`; and variable arguments handling macros.

Describing his reactions to the preprocessor defined in the draft, Banahan speculated that here the ANSI committee had used great artistic licence. Existing preprocessor code would be broken. On the positive side, Banahan suggested that now the preprocessor was better described. By a series of interesting examples, he illustrated

features of the proposed preprocessor.

The final lecture in the afternoon session was given by David Tilbrook, a veteran C programmer and UNIX guru. Tilbrook gave a historical prespective to the development of C from its early PDP-11 days to the present. Tilbrook made the point that most early C programmers were experienced professionals while today C is being used by programmers without knowledge of any other language and little understanding of the underlying machine on which their programs will run. These programmers will certainly benefit from the proposed standard.

The open meeting concluded with a panel session including all the speakers. The audience raised a variety of questions ranging from when will AT&T supply UNIX with a standard conforming C compiler to the relationship between the proposed standard C and Stroustrup's C++. The BSI organisers co-ordinated by Paul Neale received a vote of appreciation from the chair for their efforts in contributing to the success of the meeting.

A major objective of the C Panel in organising this open meeting was to promote standardisation of C and faciliate UK public comment on the draft standard. The meeting was attended by the C user community at large in commerce, industry and education as well as C compiler developers and suppliers in the UK.

Interested BSI members and members of the public can obtain copies of the current C standard draft directly from the BSI. Public comment is invited; and all comments received by the BSI will be processed by the BSI C Panel and copied to the ANSI X3J11 committee.

4. *Future Meetings in 1987*

ANSI X3J11	ISO Working Group 14	BSI IST/5/14 "C Panel"
		5 May 87
← June 87 →	Joint ISO/ANSI Meeting Paris	
		11 Aug 87
14-18 Sept 87 Boston		
		3 Nov 87
7-11 Dec 87 Phoenix		

Once an ANSI standard for C has been approved, it is likely to be put forward for registration as a Draft International Standard for C and, following review and approval, become the basis for ISO C. The ISO Standard for C would then be adopted as a BSI C standard. It would be subject to regular standard review procedures; and as long as C continues to be a "living language" subject to new developments, the work of the BSI C Panel will continue.

5. *Summary of Progress to Date and Future Timescales*

The standardisation process is essentially an iterative one; the essence being to achieve agreement between all parties — in a word: consensus. The figure below charts progress to date and milestones for the future.

ANSI	ISO	BSI
C project approved and X3J11 committee formed (1983)		Monitoring C Standard Progress
C Language Information Bulletin published for informal comment (July 1985)		
	NWI on C proposed by ANSI (Dec 85) NWI approved and Working Group 14 formed (Apr 86)	
		BSI C Panel formed (July 86)
X3J11 reach consensus on Draft Proposed Standard (Sept 1986)		
	dpANS submitted as Working Paper	
Public Review of dpANS (7.11.86-7.3.87)	Registration of dpANS as ISO DP Letter ballot 11.86	
		BSI publication of dpANS as BSI Draft for Public Comment (Jan 87) C Panel Open Meeting (Feb 87)
ANSI Standard (end of 87?)		
	Draft International Standard (end of 87?) International Standard for C (sometime in 88?) → BSI C Standard	

The iteration involved in the process consists of several loops. Within the ANSI work, there is a tight inner loop where agreement on the proposed standard must be achieved within X3J11 and an outer loop where public approval is sought. Internationally, agreement on the standard must be achieved within the C Working Group and the standard must gain approval from the member countries of ISO.

X/OPEN — What, Who, Why, When

John Tottenham

ICL

1. *What is X/OPEN?*

The X/OPEN Group is a unique consortium of eleven of the world's major information systems suppliers who have come together to agree on standards for operating systems and applications portability.

X/OPEN is not a standards setting organisation, it is a joint initiative by members of the business community to integrate evolving standards into a common, beneficial and continuing strategy. The keystone of this strategy is the common Applications Environment, a complete environment for the easy development, porting and running of applications across systems from all X/OPEN Group members.

2. *Who are the members of X/OPEN?*

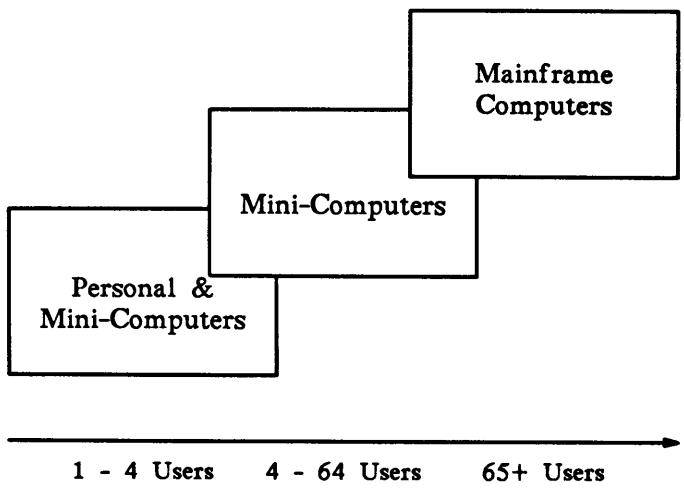
Currently, the eleven full members of X/OPEN are: AT&T, Bull, DEC, Ericsson, Hewlett-Packard, ICL, Nixdorf, Olivetti, Philips, Siemens, and Unisys. All these eleven members have made substantial financial and technical commitments and will continue to do so, providing users with a higher level of insurance for the future supply of systems based on industry standards than a single manufacturer alone could do.

In addition to the full members, numerous other companies and consultants in the Information Technology Industry have contributed to the technical and marketing programmes of X/OPEN.

3. *Why was X/OPEN formed?*

The formation of the X/OPEN Group was a direct result of two major changes in the Information Technology Industry in the early 1980's, the emergence of the Department as a large scale user of computer systems, and the growing market fragmentation caused by propriety operating systems, particularly amongst the small to medium sized mini-computers.

The three major categories of systems are simply identified according to the number of terminals attached:



This breakdown is significant since it maps the current areas where market dominant or defacto operating systems prevail, i.e. the personal and mainframe segments, and the "middle ground" when major growth was predicted but lacked any dominant operating regime.

It was this absence of a single operating system standard that was seen as a constraint on the development of the middle ground or Departmental computing market. The continuation of propriety operating systems fragmenting the market into small machine specific populations that would not attract the software industry to develop applications and hence the application software tends to be limited to that developed by the manufacturer. This means that the computer manufacturers find themselves caught in a vicious spiral with insufficient applications to expand their base and too small a base to attract the independent software industry to develop applications for it.

Recognising this problem, X/OPEN was formed in early 1984 to adopt an Industry Standard Operating System (ISOS) in the "middle ground". At that time, the only credible ISOS appeared to be UNIX, and it is the System V Interface Definition (SVID) that was eventually adopted as the first plank of the Common Applications Environment.

4. When did X/OPEN happen?

As mentioned above, X/OPEN was originally formed in early 1984, though at that time it was wholly European with the initial five members being: Bull, ICL, Siemens, Olivetti, and Nixdorf (in these early days it was called BISON from the company initials).

During 1985 Philips and Ericsson joined, and the first edition of the Portability Guide was published, making the X/OPEN standards available to the public.

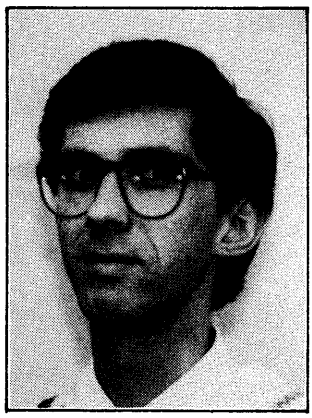
In 1986 X/OPEN gained its American members, Hewlett-Packard, Sperry (Unisys), and DEC, with AT&T joining in January 1987 at the same time as the second edition of the Portability Guide became available.

In three years X/OPEN had evolved from an idea to a practical reality backed by eleven major international information systems suppliers and with wide support from users, software industry, and government.

EUUG Tape Distributions

Frank Kuiper
...!mcvax!frankk
...!mcvax!euug-tapes

Centrum voor Wiskunde en Informatica
Amsterdam



Frank Kuiper was born in the lucky year of 1959 (a good vintage!), in Winschoten, in the north of Holland. He has a Bachelors degree in computer science and in his spare time (if he can find it), he "just goes nuts" about real-size and model trains. As young boy, he dreamed of machines with flashing lights and lots of buttons to push on. He also dreamed of sitting at a big desk, shuffling papers around and giving orders. Apart from the "giving orders" part, he thinks he has managed nicely.

He is involved in the EUUG tape distribution because "somebody told me to do it".

This is a list of the EUUG distributions, as available in September 1986. It is a general description of the available tapes. Any changes of the contents of the tapes, as well as announcements of new tapes, will be placed in eunet.general and the EUUG Newsletter.

Prices of the tapes are in Dutch guilders (DFl), and do not include VAT or postage. Note also that you have to be an EUUG member (or a member of a local UUG) to obtain tapes at list prices. Non-members will have to pay an extra DFl 300,- per tape.

EUUGD1 R6
UNIX V7 system, specially made for small DEC PDPs (11/23, 11/34, etc.). The kernel supports the UK terminal driver. V7 source license minimum.
Price: DFl 120,-

EUUGD2
Early Pascal compiler of the Free University of Amsterdam. V7 source license minimum.
Price: DFl 120,-

EUUGD3 R3
UNIX networking software, news, and some auxiliary programs. A fairly debugged version of UUCP and X.25 support will be added if a copy of the source license agreement for at least UNIX Version 7 is included.
Price: DFl 60,-

If requested, two tapes containing the major news-groups received on the Continent for the last several months are available.
Price: DFl 240,-

EUUGD4
Software tools, sampled by the Software Tools Users Group. Most of the software is written in Ratfor, for which a Fortran support tool is included. This tape is available in different formats: DEC RSX, DEC VMS, UNIVAC, IBM MVS, UNIX `tar`, MIT line feed format, and MIT card format (80 columns).
Price: DFl 150,-

EUUGD5
A collection of benchmark programs made up by EUUG.
Price: DFl 60,-

EUUGD6 (USENIX 83.1)
USENIX tape, containing contributions from various UNIX System Group Members. This is a license dependent distribution: V7, V32, SIII, V6 or no license disclosure available.
Price: DFl 240,-

EUUGD7
UNIX!STAT Version 5.2. A collection of about 25 data manipulation and analysis programs written in C by Gary Perlman.
Price: DFl 60,-

EUUGD8
A collection of useful software, based on the so-called Copenhagen tape (EUUG UNIX conference Autumn 1985). It consists of the following:

`cph85dist` The Copenhagen '85 distribution (Including all the Langston binaries).

`astro`	Miscellaneous programs about astronomical events.
`compress`	A program to compress large files.
`hack`	A game (Version 1.0.3).
`kermit`	A file transfer program. (Version 42).
`langston`	A large number of games from Peter Langston.
`macintosh`	Miscellaneous programs for UNIX to Macintosh communication.
`magtape`	Manipulation of ANSI tapes.
`mandelbrot`	Image generation program.
`mh.5`	A message handling system.
`new_curses`	Bug fixes to curses for making `wm` run.
`patch`	A program to automatically install bugfixes.
`ptxnews`	A program to generate an index of news subject.
`rand`	An editor.
`rman`	A remote manual server system.
`rn`	A newsreader program (Version 4.3).
`search`	Another game (4.2 BSD only).
`shar`	Shell archiver.
`stage2`	A compiler.

stars	A database on bright stars in our galaxy.
stat	A statistical package from Gary Perlman.
strings	A portable version of the **libc** strings routines.
vttest	Test of VT-100 emulations.
webster2	Webster's 2nd dictionary.
wirewrap	A component generator for wirewrap constructions.
wm	A window manager (4.2 BSD only).
xlisp	A **lisp** interpreter.
Emacs	latest public domain version of this editor (currently 16.60.10).
Netnews	the public domain part of EUUGD3.
(**Yacc-pcc**	By: J. A. Dain, Dept of Computer Science, University of Warwick. For **Yacc-pcc** you need at least AT&T V7 source licence. This part is only included on request.)

Price: DFl 120,-

EUUGD9

A collection of useful software, based on the so-called Florence tape (EUUG conference Spring 1986). It consists of the following:

INDEX	Indexes of articles on the net.
RFC	"Request For Comments" — proposed standards papers.
mh-6.4	Berkeley's enhanced UCI Mail Handling system .
emacs17.49	Richard Stallmans **GNU emacs** editor.
ned	The **rand** editor, version E17.
scame	An **Emacs**-like editor.
teco	The editor known from DEC systems, for VAX and 6502.
netnews	The source needed to run **news**, 2.10.3-alpha.
rn	Larry Wall's program to read the news stuff, version 4.3.
statistic	The **UNIX|STAT** statistical package from Gary Pearlman.
forth	A compiler for the forth language for VAX BSD systems.
xlisp	A **lisp** interpreter, version 1.4 by David Betz.
trc	Daniel Kary's expert system building package.
terminfo	A library of routines handling screens through **TERMCAP**.
vsh	Visual shell 4.2, for various UNIX systems and machines.
window	A windowing system.
less	A paginator à la **more** or **pg**.
rfs	The Remote File System from T. Brunhoff of the University of Denver.
rpc	The Remote Procedure Call system from Sun Microsystems.
kermit	The C-kermit transmission program version 4C(057).

langston The Langston games for 4.2 BSD VAX and Sun systems.

hack The famous rogue-like game, 1.0.3.

rogomatic The automatic rogue game.

battlestar An adventure-like game.

galaxy Yet another game.

Price: DFl 150,-

EUUGD10

MMDFIIb. Multichanel Memo Distribution Facility (version IIb). This is a powerful, domain oriented mail system with access control and the ability to communicate over a variety of network systems including TCP/IP, JANET, UUCP, PHONENET, etc. It has been ported to a variety of UNIXs including but not limited to 4.[123] BSD, 2.9 BSD and System III/V, on a variety of different hardware. You should first obtain a license agreement by sending a message to euug-tapes@mcvax. Return the signed license with your order.
Price: DFl 90,-

Ordering Tapes

If you want to order any tape, please write to:

EUUG Tape Distributions
c/o Frank Kuiper
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

For information only:

Tel: +31 20 5924056 (or: +31 20 5929333)
Telex: 12571 mactr nl
Internet: euug-tapes@mcvax (or: frankk@mcvax)

Please note that for distributions D1, D2 and D3 (and in some cases also for D8) a copy of your source license agreement with AT&T for at least UNIX Version 7 should be enclosed.

Note also that you have to be an EUUG member (or a member of a national UUG) to obtain tapes at list prices. Non-members will have to pay DFl 300,- per tape extra as handling fee. Please enclose a copy of your membership or contribution payment form when ordering. All tapes come in **tar** format, 1600 bpi (unless specified otherwise). 800 bpi is possible on request. Tapes and bill will be sent separately.

EUUG Tape Distributions Order Form

This page may be photocopied for use.

Name: .....

Company: .....

Address: .....

.....

.....

I would like to order the following:

.....

.....

.....

.....

EUUG (or national UUG) membership form enclosed? Yes / No

Copy of AT&T source license enclosed? Yes / No

"I declare to indemnify the European UNIX systems User Group for any liability concerning the rights to this software, and I accept that EUUG takes no responsibilities concerning the contents and proper function of the software."

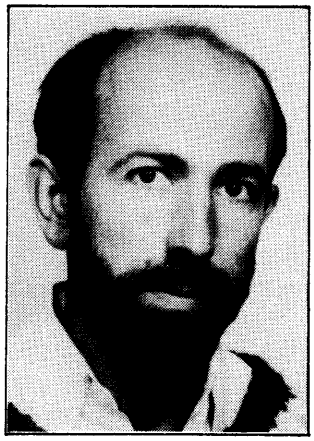
Signature .....

Date .....

EUnet

Peter Houlder
uknet@ukc.ac.uk

Computing Laboratory, University of Kent



Peter Houlder has been in the Computing Laboratory at the University of Kent for the last 30 months and looked after day to day uknet admin work in the last 18 months of that period.

He graduated in Geography from Kings College, London in 1970 and then spent 9 years in business — dropping out in 1979. He then spent a year touring Norh, Central, South and Carribean America, became interested in archaeology and spent three years excavating in Britain and Europe.

Two Masters degrees, the first in Archaeological Sciences and the second in Computer Science, followed in successive years. Maggie in the meantime reduced archaeological funding, so he arrived in 1984 kicking and screaming in the world of Computing. He has since got to quite enjoy it.

He is married with two labradors.

1. *Introduction*

This is the first of a series of articles giving information about the European UNIX network, EUnet. This particular article contains a short section on the UK network and it is hoped that network administrators in different countries will write later articles, or sections for inclusion in articles.

2. *EUnet as part of International Networks*

EUnet is the European UNIX network, which started in April 1982 at the European UNIX Systems Users' Group (EUUG) meeting in Paris. EUnet is part of the international group of UNIX based networks, which at present include ACSnet (Australia), USENET (USA), CDAnet (Canada), JUNET (Japan), SDN (Korea) and unnamed network in Israel and New Zealand. Unlike its US predecessor, which splits news and mail as two separate services, EUnet uses the same network for both news and mail. There is some confusion in terminology when referring to USENET. Officially it should be only used only to refer to the North American mail network, but unofficially it tends to be used as a term for all the international UNIX networks. The number of UNIX hosts connected to the international networks listed above varies daily, but the number of unique claimed names at present (27/4/87) stands at 9509. The backbone site for EUnet is **mcvax** in Amsterdam, which has direct links to all the other intercontinental and European backbone sites, along with direct or indirect links to many other non-UNIX based networks. Each

country in EUnet also has its own backbone site: see table below. There are also many important feed sites on the individual national networks, such as **seismo** and **ucl-cs**, which have important links to other networks. All backbone and feed sites must be capable of running UUCP, UNIX to UNIX CoPy, but it is possible for non-UNIX based sites to connect to a UNIX site for mail purposes. All backbone and feed sites must be capable of running UUCP, UNIX to UNIX CoPy, but it is possible for non-UNIX based sites to connect to a UNIX site for mail purposes. All backbone sites and some feed sites provide automatic routing, news feeding and some gatewaying between networks, but only the backbone sites handle registration. Backbone sites are important for several reasons. First they ensure uniqueness of uucp names, as backbone sites should only register unique names. Second they try to ensure protocols are maintained, by rejecting mail or warning sites that indulge in dubious mail practices. Finally they pay the bills to the various carriers and in turn collect the money to pay those bills on a network agreed usage basis.

3. *EUnet*

At present 16 countries, listed in the table below, are part of the EUnet. This is not strictly true as the Greek and Norwegian sites are only acting as backbones and the Yugoslavian site is not yet on-line. The actual size of any particular network is however difficult to assess because "hosts" may be anything from single-user machines to gateway machines for the internal networks of large multi-user organisations. The other problem is that sites may use more than one name, either because of name aliasing or the use of a local network that is not hidden to the outside world. The terms "site" and "host" are both used to refer to individual mail-handling machines connected directly to EUnet national networks.

EUnet as of 27th April 1987				
Country	**Hosts**	**Names Used**	**Backbone**	**E-mail**
Austria	19	27	tuvie	tuvie!plank
Belgium	11	29	prlb2	prlb2!ml
Denmark	38	42	diku	krus@diku
Eire	9	10	einode	einode!simon
Finland	45	45	tut	tut!hmj
France	68	68	inria	inria!devill
Great Britain	208	248	ukc	uknet@ukc.ac.ukc
Greece	4	8	ariadne	ariadne!kostas
Iceland	1	1	hafro	hafro!gunnar
Italy	25	25	i2unix	i2unix!roby
Netherlands	93	159	mcvax	piet@cwi.nl
Norway	6	7	nuug	kvvax4!franki
Sweden	123	146	enea	enea!ber
Switzerland	29	29	cernvax	dietrich@cernvax
West Germany	107	107	unido	ap@unido
Yugoslavia	1	1	yupiter	yupiter!root
Total	787	952		

3.1 *Uknet as part of EUnet*

Great Britain started its EUnet links back in 1982, but a fortuitous short term link, via an ex-student, directly to USENET in the USA, meant that its close involvement

really began in 1984. In early 1985 it had some 29 sites all connecting directly or indirectly to the Computing Laboratory at the University of Kent, hereinafter referred to as **ukc**. The creation and continuation of the network is almost entirely due to Peter Collinson, who had the necessary UNIX know-how and contacts to get the network started. However Sean Leviseur, Richard Hellier and in the last 18 months myself have all helped with support software and day-to-day administrative work. The first sites to join the network were predominately academic or commercial sites with close academic affiliations. Later growth has however been fairly evenly split between academic and commercial sites. In the first three months of the network the number of sites doubled to some 60 sites. By the end of 1985 this number had increased to 80 sites. Since that time the number of sites has grown by a steady 8 sites per month, and now stands at 209 sites (27/4/87). The growth in the network shows no sign of flattening off, so if this continues in the foreseeable future approximately 100 extra sites can be expected to join annually.

4. Further Reading

An excellent article on international networks called "Notable Computer Networks — John S. Quartermain and Josiah C. Hoskins" appeared in the October 1986 edition of The Communications of the ACM. Some of the above information has been gleaned from this article.

UNIX Clinic

Nigel Horne
njh@root.co.uk

ROOT Technical Systems

Nigel Horne has worked solely on UNIX since graduating in 1980 from Westfield College, London (and to a certain amount as an undergraduate as well). He has been involved in UNIX from the early days of "real" UNIX, the days of `seek()`, `roff`, PDP11's (they didn't even have split I+D in those days), keys for typing in the bootstrap, through to today when there are System V, 4.3 BSD, industry standards, and just as much confusion as when it all started.

Nigel is now a Director of Root Technical Systems.

It is hoped that this page becomes a regular feature in future EUUG newsletters. The idea is to start a forum of discussion and trouble shooting, on all aspects of using the UNIX system. Whilst many of the questions may well be slanted towards the beginner, it is hoped that there will be something of interest for just about everyone in the column.

You can send questions to me either via EUUG, by direct mail or even using electronic mail if your machine is connected to EUNET either directly or via another machine. If you want to try sending mail electronically try both of the following commands: if neither of them work, it is unlikely that your machine is connected to EUNET.

```
mail mcvax!ukc!root44!njh
```

or

```
mail njh@root.co.uk
```

I'm sorry that I can't enter into any discussions about advice given in this column, and any material sent to me by any of the means above will be deemed to be acceptable for publication.

As an introduction I thought I'd cover two questions in one by covering a question that recently showed itself to me. I've slightly doctored it for this example and no names are mentioned to hide certain peoples' identity. The problem manifested itself on a PC/AT look-alike running System V Release 2. Everything was in order except when we came to use the supplied screen editor `vi`. All that the editor did was to print the message `memory fault -- core dumped` and leave the terminal in a strange mode — without `echo` and the such. The problems here were: why didn't `vi` work, and how do we get the terminal back into a sensible state? Answering the second question is easy. The terminal was left in so called "raw" mode, which meant no `echo`, no backspace facility, and the system no longer accepted carriage-return as you'd expect. The cure is simple: first type control J. Why? Ah well, UNIX actually takes control J to mean end of input, not carriage return; however it

normally maps one on to t'other, so typing control J just clears any junk characters in the input buffer. After doing this, type

```
stty sane<^J>
```

making sure to use control J again. This brings the terminal and keyboard back into a "sane" state, that is with echo on, carriage return accepted, and so on.

The problem of the core dump? This took some looking for. No other programs on the machine acted in this way, and we began to suspect that our copy of the image of `vi` on the hard disk was corrupt. In fact it was far simpler, we were using a PC/AT with 512Kb of RAM. `Vi` needs more RAM than this to enable it to run (remember that a fair proportion of the 512Kb is taken up by the UNIX operating system image), and instead of exiting gracefully with a *need more core* message and returning the terminal to a sane state, it just crashed. Solution? Buy more memory.

I hope to hear any questions about UNIX that you may have in the near future. I regret that I cannot answer questions about which hardware to buy, or that I may not cover all the questions I receive, but rest assured that I will try to acknowledge all material I receive.

Review of IEEE Trial-Use Standard Portable Operating System for Computer Environments POSIX+

Cornelia Boldyreff
...!mcvax!ukc!reading!uoseev!corn

Department of Electronic and Electrical Engineering
University of Surrey
Guildford, Surrey GU2 5XH

The draft standard published by the IEEE for comment and criticism was issued in April 1986 with the proviso that its distribution for comment shall not extend beyond one year. In order to facilitate wide-spread distribution, the standard is available from the IEEE and ANSI as well as Wiley-Interscience. Its purpose is to define a standard operating system interface and environment based on the UNIX operating system. Primarily, its focus is the C language operating system interface required to support portable applications at source code level. Similar issues are addressed by the AT&T publication, System V Interface Definition (SVID). The SVID addresses source-level interfaces across AT&T's UNIX System V product; however, unlike the SVID, the POSIX standard is not a specification of a commercial product. The X/OPEN group of UNIX manufacturers has also defined a similar UNIX applications interface, the Common Applications Environment (CAE), based on the SVID, their principal aim being to ensure software portability of UNIX-based commercial products. Like the SVID, CAE is tied to the AT&T UNIX product. X/OPEN has expressed its long term support for the POSIX standard, and AT&T's SVID states that conformance with the IEEE standard will be "strongly considered" after its formal approval.

There are three major components to the standard:

- Definitions — this initial chapter deals with terminology used through the standard, general concepts are described informally, and various symbolically named variables and constants are defined.
- System Interface and Functions — this forms the core of the standard. These chapters define a C Language Binding for Process Primitives and the Process Environment; Files, Directories and File Systems; Input and Output Primitives; Device- and Class-Specific Functions; and Password Security.
- Key Interface issues — Portability; Media Formats; and Error Handling and Recovery.

Currently, the POSIX standard does not address the user interface and associated commands; graphical interfaces; DBMS interfaces; record I/O; or object or binary code portability. Since publication, the P1003 Working Group has formed in addition two new groups addressing the shell and tools interfaces and conformance testing:

- P1003.2 — The shell and tools facilities
- P1003.3 — Verification test specifications

POSIX explicitly does not provide recommendations for an end-user interface; the recently formed POSIX subcommittee is concerned with shell and tools facilities from

the standpoint of syntax and services that an applications programmmer might wish to access via the popen or system function calls. There are other groups concerned with defining a User-Interface for Applications: the X/OPEN group and the ECC ESPRIT PCTE. The latter tests are required because there is a Federal Information Processing Standard targetted for POSIX in the USA.

The IEEE Working Group formulating the POSIX standard includes staff from all the major US computer companies in the UNIX market. Those at a recent meeting included staff from Amdahl, Apollo, AT&T, Charles River, Concurrent Computer Corp, DEC, DG, Gould, IBM, Interactive Systems, H-P, P-E, Sperry, Sun, Tektronix, and TI. UNIX user groups (USENIX, X/OPEN /usr/group) and US government and military users are also represented on the Working Group; and there has been some participation from outside the USA including British members. British companies active in reviewing the POSIX standard include British Airways. In the Acknowledgements concluding the P1003.1 text, over 200 organisations are thanked for their contributions to the Working Group.

The POSIX standard is closely related to two other standards: the 1984 /usr/group Standard and the Draft Proposed ANSI Standard for C formulated by the ANSI X3J11 Committee. The /usr/group Standard work has been subsumed by IEEE P1003's POSIX Standard and ANSI's X3J11 C Standard work. P1003 has left the definition of library functions required for a C implementation in any environment to X3J11; that is, POSIX refers to the C Standard for these. The C standard in turn does not define operating-system-specific functions, leaving these as the province of the POSIX P1003 Standard. There is active liaison between the P1003 committee and the X3J11 committee who over the past few years have developed a good working relations.

This Trial Use Standard is expected to become a full-use IEEE standard and an ANSI standard within two years; ANSI in turn have proposed to ISO a New Work Item based on the P1003.1 effort. Their aim is to facilitate international participation in this work leading to its adoption as an international standard.

In its present form, POSIX does not provide a "functional" specification of a portable operating system independent of any specific language, i.e. in this case C, binding.

POSIX is a much needed effort to standardise an existing applications environment interface based on UNIX systems and complementary to the C standard work. If it could be made to assume this more generalised functional role, then it could well become the basis for related standards work in the area of Open Systems Interconnection.

EUUG NATIONAL GROUPS

AFUU (France)
c/o SUPELEC,
Plateau du Moulon,
91190 GIF-SUR-YVETTE,
France.

BUUG (Belgium)
Department of Medical
Informatics,
VUB, Laarbeeklaan 103,
B-1090 BRUSSEL,
Belgium.

DKUUG (Denmark)
Kabbelejvej 27B,
DK-2700 BRØNSHØJ,
Denmark.

EUUG-S (Sweden)
NCR Svenska AB,
Box 4204,
17104 SOLNA,
Sweden.

FUUG (Finland)
OY Penetron ab,
Box 21,
02171 ESPOO,
Finland.

GUUG (Germany)
Mozartstrasse 3,
D-8000 MUNICH 2,
Federal Republic of Germany.

IUUG (Ireland)
Glockenspiel Ltd.,
19 Belvedere Place,
DUBLIN 3,
Ireland.

ICEUUG (Iceland)
University Computer Centre,
Hjardarhaga 4,
Reykjavik,
Iceland.

i2u (Italy)
Viale Monza, 347,
20126, MILANO,
Italy.

NLUUG (Netherlands)
Xirion bv,
World Trade Center,
Strawinskylaan 1135,
1077 XX,
AMSTERDAM,
The Netherlands.

NUUG (Norway)
Unisoft AS,
Enebakkvn 154,
N-0680 OSLO 6,
Norway.

UKUUG (United Kingdom)
Owles Hall,
Buntingford,
Herts. SG9 9PL,
United Kingdom.

UNIGS (Switzerland)
c/o Instutut fur Informatik,
ETH Zentrum,
8092 ZURICH,
Switzerland.

UUGA (Austria)
TU Wien,
Inst fur Praktische Informatik,
Gusshausstr 30/180,
A-1040 WIEN,
Austria.

EUUG

**The Secretariat: European UNIX® systems User Group, Owles Hall, Buntingford,
Herts SG9 9PL, UK. Tel: Royston +44 (0) 763 73039 Facs: Royston +44 (0) 763 73255
Network address: euug@inset.uucp**

The Secretary
European UNIX® systems User Group
Owles Hall
Buntingford
Herts. SG9 9PL.
Tel: Royston (0763) 73039.