EUUG

EUROPEAN UNIX® SYSTEMS USER GROUP NEWSLETTER

Volume 4, No. 2
SUMMER 1984

EUUG
European UNIX† Systems User Group

Newsletter Vol 4 No 2

(Late) Summer 1984

† UNIX is a Trademark of Bell Laboratories.

EUUG CONFERENCE AND EXHIBITION

DATE FOR YOUR DIARY

The next EUUG Conference and Exhibition will be held in Paris on the following dates.

SPRING 1985

Venue:	Palais de Congress — Paris — France
Conference:	1 - 3 April 1985
Exhibition:	2 - 4 April 1985

Further details will be included in the next EUUG Newsletter

●

Is This System Mannerist or Rococo in Style?

Salt Lake City and USENIX

P. Rayonnant

Sunday

Five o'clock, the new frontier beckons. Goodbye Dick, goodbye Doughball; don't eat any cables while I'm away. Why isn't the bus here? What, they changed the timetables last week? And this is a holiday anyway? Taxi. No, I don't know which travel agency ordered the tickets, don't you have them indexed by traveller-name? "For operational reasons we are flying this morning in a Bandicoot 20 seater across the North Sea. I'm your co-pilot, and once we're airborne, I'll be serving coffee".

London - yick. Gorky Park - the book was better.

I visited my first two U.S. airports, and got lost in both. There seems to be an assumption that the traveller knows which airline he is flying with. For the connecting flight from St. Louis to Salt Lake City my ticket just said WA691. Each airline has its own monitors, departure lounges, etc. and there is no information desk. Could be *Wright Bros. Airlines*, I suppose. If in doubt, ask a policeman.

Had a beer while waiting for the flight. Not promising.

Flying across the central U.S. with *Western Airlines* was dull, the country is flat, just like The Netherlands, except no one has bothered to make the waterways run in straight lines. How untidy.

In Salt Lake City I just followed the signs to "Bagage Reclaim". Of course, there are two....

This hotel is IMPRESSIVE. Must be old, the lifts are just like in The Netherlands, *slooooooow*. The single-room has a bed wider than it is long, three pillows wide, a Mormon-bachelor occasional-double? Speaking of Mormons, the Hotel Utah is right next to the centre of Salt Lake City, where the Temple is (I can look down into it from my room); the streets in Salt Lake City are not named, but numbered on a Cartesian grid ("100 South 300 West"), and yes, Temple Square is at 0,0.

The clock says eight in the evening, the body says "It's almost time to get up tomorrow", so it must be time for a wander around (short, everything is closed), and something to eat. "Hi *Sir!* Have you made your choice yet?" Mumble mumble. "Thank you, *Sir!* Will that be all?" Mumble mumble. "Thank you, *Sir!* My name's Larry, and I'll be your waiter for this evening." Mumble mumble ZZZZZZZZZZZZZZZZZZzzzzzz.

Monday

No! Not yet! Not Yet! OK. I suppose so. Lunchtime. Wandered around. Still doesn't look very alive (I don't either I suppose).

Registration opens in the late afternoon, so I get registrated. Along with the registration packet comes the proceedings, a really handy thing to have *before* the conference; now I can ask even more interesting questions since I'll know it all beforehand. Why did I bother coming? It also means that I don't have to write everything down in a boring trip report, I only need to give the undocumented features not in the manual.

Finally bump into some people I know, time to eat and drink. Well, more eat than drink, Utah has some strange laws about the sale of alcohol. Fortunately beer is easy, but were we foolish enough to want wine or so with our meal, the restaurant has a *Mini State-Liquor Store* at the front (within and part of the restaurant). Customers who want wine have to go there themselves, buy the evil stuff and return to their seats. The restaurant is allowed to provide glasses and open the wine for you, but is *not* allowed to pour it, you have to do that yourself. Weird.

Back to someone's hotel room for a wine tasting, and very nice they tasted too.

Tuesday

Wandered around some more. Saw *Star Trek III - The Search for Spock* in the afternoon, for want of something better to do. I was looking less like Spock by time time I came out.

There were some tutorial sessions on *C-Style*, *4.2BSD Internals*, *Networking*, etc. Some were sold out, some had a handful of attendees. USENIX seems to find these tutorials popular, and wants to continue them at future meetings. The problem is finding people who know the the stuff *and* can communicate it well. Suggestions are welcomed.

The exhibition was also under way, less than 100 stands I would guess. Nothing really new. AT&T wouldn't let anyone touch their machines. Some of the 68K boxes are becoming quite nice; Integrated Solutions were claiming theirs was "what the Micro-Vax should have been".

```
<<`echo "Goodbye World!"`
```

to the Bourne Shell was popular amongst the vendors I tried it on, some machines just crashed, others needed to be powered off.

The most interesting thing about the exhibition was the Ethernet. There was an Ethernet cable to which vendors were invited to attach their machines. I saw about six different brands of hardware all chatting away nicely (using TCP/IP protocols), including a laser printer station from Imagen. Apparently there was hardly any preparation for this Ethernet demonstration, all the more remarkable.

Tuesday Evening

Things are beginning to move now, people are beginning to arrive. And the hospitality suites are open!

Hospitality suites are places where, under the influence of free food and alcohol, you are scrutinised and pestered by representatives of the company supplying the food and drink with a view to them weaning you away from your current blissful employment.

But wait! There is something wrong! Here I am, eating and drinking in a hospitality suite, *and the reps are ignoring me!* What is wrong? Why is everyone looking at the T.V. set? I ask, and it seems there is a *basketball* game on, the last one of the season or something. It looks exceedingly dull, everyone runs to one end of the pitch, the ball may or may not go through a hoop attached to a board, then everyone runs back to the other end; the team which loses seems to be the one which fails to put the ball through the hoop most, as far as I could see each team gets an equal number of tries. If it was profiled, all the running around would probably be moved out of the loop‡.

Other hospitality suites were the same. Refreshing (at the time).

Wednesday Morning

Come Wednesday and it is time for the official conference to begin in earnest. The Symphony Hall is impressive, and I found the lack of central aisles (one had to enter from the sides) adequately compensated for by the comfy chairs and leg room.

‡ This wasn't the only strange game saw on the television, there was also *American football* and *baseball*. American football is remarkable in that only one person on each team ever actually uses a *foot* to kick the *ball*, and he spends most of his time sitting on a bench, waiting for the ball to be placed in front of the goal posts. Maybe we should rename soccer *handball*, since only one person on each team is allowed to handle the ball?

The Dutch for baseball is "honkbal".

There was a professional audio-visual set-up, three large screens at the back of the stage, one for overheads (hardly used), one for slides, and the central one with a projection of the speaker's head and shoulders. Hello Mum. Slides were definitely the order of the day, and some poor soul slaved away night and day making slides from the overheads which everyone brought; he deserves some thanks.

Some computer-graphics films were shown on the multi-media equipment as the UNIX community filtered in for the start of the meeting (why do these things always start so early?). Someone told me they were from the last SIGRAPH. Econo-Mars Earthtours looked preferable to British Caledonian.

First business of the day was some announcements. There were two competitions being run during the conference, the First USENIX Computer GO Tournament, run by Peter Langston, and the UNIX Trivia Quiz, run by Rob Pike, 84 questions along the lines of "What was the telephone extension of the inventor of multiplex files?".

The next meeting will be held in Dallas (there will be a simultaneous conference and trade-show held there by /USR/GROUP, and future meetings will be in Portland (USENIX only), with Atlanta, Boston and San Francisco as possibilities after that. USENIX are also sponsoring limited attendance (approx. 120) workshops, on topics such as distributed systems and computer graphics; participants will have to send in a position paper first.

The USENIX Newsletter *;login:* now has Brian Redman as technical editor, and USENIX are still looking at the possibilities of a journal and electronic publication.

After that, someone from the EUUG spoke for five minutes about what was happening in Europe and announced the forthcoming EUUG meeting in September in Cambridge. One of his slides appeared to be in Dutch, but it was just back to front and upside down.

There then followed the keynote address, *An Architecture History of the UNIX System* by Stuart Feldman (the man who brought you *make*). This was a real treat. It didn't appear in the proceedings, as that would have spoiled the fun. As it was there was lots of fun, not all of it expected. The talk was basically a pictorial trip through the last thousand years of Western architecture, detailing the rise and decline, with side references to the history of the UNIX system. The added fun came as Stuart's slides caused havoc with the slide-projector, having the audacity not all to be of the same thickness (many were borrowed from libraries and museums). At one point, as one refused to be ejected from the projector, there were cries to pass the remaining slides round the audience instead; coffee was taken early as the projector was dismantled.

Michael Tilson then gave a well-reasoned talk about the need for standards in UNIX, and the coming confrontation between UNIX and Godzilla. He gave a history of UNIX, including a slide depicting the real UNIX family tree, it looked like a plate of spaghetti.

Hot in pursuit came some talks about electronic mail and news. The most interesting was given by Robert Elz and described ACSNET, the Australian alternative to UUCP. There are some published papers on this, and it is quite interesting, all routing being taken care of by the network, explicit routes don't need to be given. There was a lot of speculation as to how this would scale up to a network the size of the current UUCP network.

Other talks in this session were *Broadcasting of Netnews and Network Mail via Satellite*, *The Berkeley Internet Domain Server*, *MMDF II: A Technical Review* and *DRAGONMAIL: A Prototype Conversation-Based Mail System*.

I didn't see all of these, having to go to a meeting over lunch, where I had a banana-split King Kong would have had trouble with.

Wednesday Afternoon

Apart from the effort needed to digest the world's largest banana split, the afternoon was spent listening to talks about distributed things.

The first session was about networks and distributed systems. In one talk about converting the BBN TCP/IP software for 4.1BSD to 4.2BSD, the claim was made that the throughput was comparable. Someone from the audience pointed out that the particular Ethernet controller being used was in fact the limiting factor in performance in both cases.

Other talks included how to synchronise the clocks of the different machines on a local area network, the *LOCUS Distributed UNIX System*, and *Project Athena*.

This last is interesting, as it gives a sense of scale of investment being made in the U.S. by computer manufacturers. Project Athena is a joint venture by MIT, DEC and IBM, which will install approximately 1600 VAX-based personal computers at MIT for an experiment in undergraduate teaching; about 2500 machines will be installed in all. 4.2BSD is the current base software. You can imagine the problems of networking, backup, maintenance and just plain scale. The results, whatever they are, will be interesting.

The second afternoon session (after the delights of fishing cans of strange liquids out of enormous buckets of ice; what is "root beer", it tasted of toothpaste?) was about distributed filesystems. Peter Weinberger talked about the *Edition 8 Network Filesystem*.

Peter was fun to watch, he bounced around so much, dashing over to try to read his own slides, that the audio-visual projectionist had to work up a sweat. The talk didn't differ a lot in content from that he gave in Amsterdam in 1982, but he'd had the time to replace the hand-written overheads by BTL-style slides which even he couldn't read.

The filesystem gives you network independence by using asymmetric virtual circuits, the client reads the server's files, the server handles permission translations and time differences. Remote objects are filesystems, not disks; pairs of machines negotiate a connection. Don't expect much from the network, as that's what you get.

The implementation uses user-level servers, and switches at the inode-level in the kernel to test if this is a remote file or not (doing it at the user library-level is described in the abstract as a "maintenance nightmare").

The other two talks in this session were also quite interesting, a system called IBIS from Purdue which is based on 4.2BSD and uses the *host:filename* syntax to access remote files, and the *Livermore Interactive Network Communication System (LINCS)*. This runs in a *very* heterogeneous environment, and includes TCP/IP connections from VAXes running VMS/EUNICE to "real" UNIX machines.

The last part of the afternoon was taken up by a panel session on distributed file systems. Not a lot interesting was said. Bill Joy was in "Paris mode".

Wednesday Evening

Back to nature, a bus to a ski-lodge in the mountains, and the official conference reception. Sunshine, trees, snow, rushing streams, all to be experienced as you stand beside a bus with burned-out transmission.

But it was fun. Cable-car from the ski-lodge (Snowbird) to the top of a mountain (11,000ft), slither around in T-shirt and gutties. Back down for food and wine (altitude affects both your breathing and also the ease with which alcohol affects your system). Crossed-legs bus ride back to the hotel, then on to the DEC hospitality suite, it was open every night, all the time. This time people in uniforms came to break it up. What fun.

Thursday Morning

Nooo problem! Breakfast is though. "What you got?" "Oh! We have ... etc." "Got any muesli?" "What's *that?*" Kellogs wins the day.

My enthusiasm for cornflakes is matched by my enthusiasm for talks about C compilers and other programming language issues. There was a talk about Modula-2 in the system-programming environment which didn't look to bad. However, many people's vocabulary doesn't extend beyond the third letter of the alphabet.

Coffee break. Well, they call it coffee; there's more strength in the C-compiler's type checking. There are also strange sticky pastries, guaranteed to necessitate the application of a Swiss Army knife to the moustache.

A number of talks about programming environments and windowing systems. Rob Pike described a development which we can't get of something he talked about before which we still haven't got. Pity, it's probably what I want.

A team got together over lunch to discuss questions such as "Which university stole UNIX by telephone?". I like pastrami. They disagreed over some of the finer points, and two different entries went in to the "Pastrami Trivia Quiz".

Thursday Afternoon

Into the kernel. Multiprocessor UNIX, what you need to do to make it run on a 3B20, the system can be configured to run in uni-processor mode (no performance penalty) or multi-processor mode (70% throughput increase).

Richard Miller, of porting UNIX to the 7/32 fame, talked about adding demand paging to System V. Only ps(1) need be changed, the performance is equivalent to the swapping system. Uses a working-set page replacement policy as opposed to the 4.?BSD global clock algorithm. It wasn't done on a Vax originally, though.

After more fishing around in buckets of ice for cans of refreshment, the talks resumed. Tom Killian gave his talk about processes as files, which Andrew Hume gave a very brief presentation of in Nijmegen. It really is neat the way it fits in with "the UNIX way of doing things". Debugging and ptrace just fall out naturally. Implementation goes through Peter Weinberger's Edition 8 filesystem, where there is a *file system type* for each of these weird things.

Thursday Evening

Time to hit the hospitality suites again. Silicon Graphics have their flight simulator in a hotel room, but the room is simulating the Black Hole of Calcutta. Still, they did have some *Anchor Steam Beer*, positively tasty compared to the competition. Have something to eat instead, visit the GO competition (won by a ported IBM-PC program), visit the Human Computing Resources Coffee and Liquers, drop in on the DEC suite on the way home, finally to bed. It must be the jet-lag.

Friday Morning

Slightly more of a problem. Cornflakes, weak coffee are beginning to take their toll.

Performance analysis and comparisons is what we have first. Someone at AIM collected a lot of benchmark data, then worked out how to display it; what it means is just the same as any other UNIX benchmark you've ever seen. Not a lot.

Two talks, one from a mannerist comparing SV to 4.2BSD, and one from a rococoist about how to speed up 4.2BSD. A big fight was expected, involving a somewhat partisan audience, but it never happened. Everyone was amazingly appreciative of both sides of the coin, or had also been to the DEC hospitality suite. Both systems have their good and bad points, you pays your money, you get the same performance provided you matched the system to the use.

Before the coffee break, Rob Pike announced the results of the Trivia Quiz. He was surprised about how much people out there actually knew about the dark history of UNIX. However, the winning entry from a team only managed 60 correct answers out of 84. The second placed entry managed 56 correct, and received the best prize, a DECtape labelled "11/20 8K sys with RP11 units July 5/72".

The talks in the remainder of the morning were missed due to a combination of worrying about an export licence for the tape I'd just received, and the need to take an early lunch with some people who had to leave the conference early.

Friday Afternoon

One of the interesting things about the talks throughout the conference was the number which dealt with forecoming technology, more saying "what are we going to do with this when it arrives?" rather than "here's my solution to last years problem". And two things which are coming are optical disks and the need to distribute software as automatically as possible where possible. I wouldn't say I agreed with the possible solutions offered to these two particular examples, but at least some people are looking a bit further than yesterday.

That was more or less the conference, and a good one it was too. There were some other talks and panel sessions, but (and this is my only complaint about the conference organisation) they were held in another place, about 7 minutes walk away and in parallel with the other talks. This made it impossible to see whatever was best at the time. Some looked like they might be interesting, the future of BSD, the UUCP/USENET projects (doomed to failure in my humble opinion) and the ANSI C standard draft.

Throughout the conference Rob Pike was taking pictures of people for the face-server, which would be digitised into high-resolution form, then digital faces of any size are generated on demand; some commonly used ones, such as the 48x48x1 faces for mail announcement are precomputed.

There was strong competition for the "hairy knees of the conference" award, Salt Lake City being a warm place. However, after a day spent skiing in his shorts, the twin pillars of flame owned by the runner-up from the Nijmegen conference, Andrew Hume, made all other entrants pale in comparison.

Friday Evening

There was a "wizards" party up in a cabin in the mountains. Fun trekking through the snow and trees to somewhere where the door was a window, the fridge was a snowdrift at the backdoor and the toilet a gopher (or similar beast) hole.

It was great, especially trying to get down in the dark through a snow covered forest to a car after the snowdrift was empty.

Saturday Morning/Afternoon/Sunday

Somewhere in here I negotiated two U.S. airports without mishap (pity British Caledonian can't learn to do that too).

What's the collective name for a bunch of U.S. kids? A brace. No wonder they all need psychoanalysis in later life, having to go through puberty with scaffolding on their faces.

Somewhere in here Saturday became Sunday very quickly. No, "Reuben, Reuben" is not about sandwiches.

"I climbed on the back of a giant anchovy, and flew off through a gap in the clouds...."

The Arlo System

David M. Tilbrook

Imperial Software Technology

ABSTRACT

Arlo is a system for constructing interactive applications on a UNIX system. It allows a new application to be developed with relative ease, whether starting from scratch or employing existing tools and components, and promotes a consistent and helpful user interface to the complete application.

This report describes some of the more important Arlo concepts and tools.

Introduction

The Arlo system facilitates the construction of interactive applications systems and command interpreters. It has been used as an interface to the UNIX shell, as a foundation for computer assisted learning systems, for rapid prototyping, as a menu system, to front end existing applications, to integrate existing applications that would not normally work together, as an interactive dialogue manager, and for a variety of other applications and packages.

At Imperial Software Technology (IST), it is being used for both the framework of the Integrated Project Support Environment and the development environment for constructing that system. Our guest login accounts frequently use Arlo scripts as the login shell (e.g. the games script); and new users are added to the system using an Arlo script. Most of our users prefer mail system written in Arlo to any of the available alternatives. The only interface provided to our Ada programming course students is an Arlo script that prohibits them from doing anything other than create, edit, compile and run Ada programs in their own directory. Arlo scripts exist to maintain our software inventory, UUCP and software releases. Further scripts check and maintain reports, interface to data bases, provide an interface for the system engineers, and run interactive games.

The Arlo Approach

Arlo provides a mechanism to deal with what, in many applications, is the dominent problem, that of 'system integration'. By system integration in this context we mean constructing a system from both new and pre-existing components and presenting the user with a consistent interface to that system. However, the scope of Arlo's concerns goes beyond that of a single static application. With continued use of Arlo it is possible to gradually evolve individual applications to meet the users' growing needs, and to achieve a consistent style of user interface across a wide range of applications.

Arlo has evolved over a number of years and changes and enhancements have been motivated by its use for applications as far ranging as telephone receptionist systems and Adventure type games [1]. However, a number of principles have been consistent throughout its evolution.

[1] The telephone receptionist system had interfaces to a small voice and data contention switch and could emulate all the actions normally performed with a telephone console. The telephone operator had only a headset and a terminal (a dialer wasn't required). The Arlo program allowed the operator to answer or initiate trunk or intercomm calls, to create or give mail messages and to do call announcing via the intercomm or the user's terminal.

The adventure game was a reprogramming, in Arlo, of Peter Langston's Castle game. The adventurer wanders through a forest and castle solving problems and mazes. At one point he/she has to login to a UNIX system and

1) The system must enable the programmer/user to create and modify interactive applications quickly and easily while ensuring that all such applications are consistent with respect to the user interface, are self documenting, and, most importantly, are easy to use without imposing unacceptable behaviour on the sophisticated user [2].

2) Applications should be encapsulated in a single self-documenting interface.

3) The support for iterative program development and testing, at low cost, must be a fully integrated part of the system.

4) The system developer must be able to ensure complete protection of both the user and the system. For example, it should be possible for the programmer to shield the user from some of the confusing error messages that some applications generate.

5) The system must be able to preserve and protect the user's history and context and be able to use that information to the user's advantage.

6) The system should not depend on any special characteristics of the user's environment or terminal, but should be able to take advantage of such characteristics if available. For example, it should allow (but not require) use of pop-up menus on a bit-map if they are available.

7) The system should be able to run, unchanged, on any reasonable size of machine [3].

So, given these principles, what is Arlo?

In one sense, it is an interactive application generating system. Part of it is an interpreter, which has interfaces to the user, to the host file system, to the host utilities (including the command interpreter), and to Arlo 'script' data bases, which are indexed data bases of Arlo 'programs'. The other parts of the Arlo system are utilties to create and validate Arlo 'scripts' or to provide documentation for the Arlo application programmers and users.

Another way Arlo can be viewed is as a command interpreter. It prompts the user for a command, the first word of which should name an item in the script data base (i.e., a 'command') which is then extracted and interpreted.

For some applications, it can be viewed as a 'Menu' system. It provides facilities to present the user with a list of available commands from which the user may select.

It can also be used to construct dialogue systems, question and answer sessions, and other such applications. It is intended to investigate the possibility of using Arlo as the target language for interactive application design systems, such as Rapid's transition state diagrams.

Classification of Arlo is difficult, since it is a versatile system. The rest of this paper attempts to describe and demonstrate some of its facilities and power.

Overall System Architecture

The following sections describe the components of the Arlo system and the Arlo programming language and data base. This report is not intended as a reference document. Thus, none of these descriptions is very detailed.

More information can be found in the Arlo Tutorial and the manual sections: Arlo(1), Arlopp(1), Arlohdr(1), Arlolog(1), Arlo(5), and Xarlo(1).

---

execute a variety of UNIX commands (e.g., launch missile and write doctor) some of which Arlo interprets by invoking the normal Shell.

[2] The over worked phrase 'user-friendly' will not be used except in this footnote. That phrase is frequently used to mean that the system can be used by absolute novices immediately. However, just as frequently it is the case that after a few uses, the system becomes a pedantic and slow-witted bore and not something with whom most of us want to be 'friends'.

[3] Arlo has been run on small address machines (PDP11/44s and PDP11/34s with overlaying), and has been run on 68000s and 3B2s. The Arlo interpreter on 4.1BSD is about 45k bytes.

The following diagram illustrates the components of the Arlo system and their relationships:

```
+---------------+                       +---------------+
| User          |       +-------+       | Host file     |
| Interface     |<----->|       |<----->| System        |
+---------------+       |   A   |       +---------------+
                        |   R   |
+---------------+       |   L   |       +---------------+
| Script and    |------>|   O   |<----->| Host command  |
| Item data base|       |       |<--+   | Interpreter   |
+---------------+       +-------+   :   +---------------+
    ^       :               :       :
    :       :               :       :
+-------+   :   +-------+   :       :   +---------------+
|Arlopp |   +-->|Arlochk|   :       +-->| Msg interface |
+-------+   :   +-------+   :           +---------------+
    ^       :               :
    :       :               :
+-------+   :   +-------+   :   +-------+       +-------+
|Source |   +-->|Arlohdr|   +-->|logfile|------>|Arlolog|
+-------+       +-------+       +-------+       +-------+
```

1. The Arlo Interpreter

The Arlo interpreter interacts with the user, prompting him/her for input. For one class of prompt (the selection prompt) the user's response is interpreted as a command. The first word of the command names or is mapped to an item in the Script and Item data base. That item is then retrieved and interpreted. The items will normally consist of one or more Arlo statements which may invoke shell commands, prompt the user for other responses, interact with the host file system, change the script data base, etc.

2. The User Interface

Normally Arlo interacts with the user via the standard input/output provided by the terminal. No special processing or modes are required. However, the system can be and has been interfaced to other terminal types (e.g., bit-map terminals) and mechanisms are provided to do pop-up menus or alternative command and response selections.

Arlo prompts for user input fall into three groups: command selection, a pause for pagination, or a prompt for a value or choice. The latter class is further divided into a prompt for a string value, a prompt for a numeric value, a prompt for a yes or no decision, a prompt for a selection from a list, and a prompt for a debugging command [4]. The different prompt classes are indicated by the prompt suffixes, as shown in the following table:

SUFFIX	Prompt for ...
]	command selection
:	string value
⊃	numeric value
(y/n)	yes or no answer where yes is default response
(n/y)	yes or no answer where no is default response
}	selection from list

[4] Debugging facilities are an important and integral part of the Arlo system. The debugging facilities are fully integrated into the programming language as keywords of the language. Facilities exist to set break points, examine variables and items, control traces, change and restart scripts, and to override protections and prohibitions. The latter facility is password protected.

>	debugging command at a break point
<CR>	carriage return

All the prompt classes support user aids in that the user may ask for a further explanation of the question by responding '?', and may ask for an explanation of the prompt class by responding '??'.

3. The Script and Item Data base

An Arlo session reads items from a preprocessed 'script object' file. Such a file can be viewed as a 'compiled Arlo program'. The file contains the script's items, plus information to access and name the items. Script object files are prepared by Arlopp.

4. Arlopp

The script file used by the Arlo interpreter is created by Arlopp, which transforms the Arlo source into a binary format file that contains the items in relocatable form, an item table, the list of item names, the name of the original source file, and other script processing information.

If Arlo is invoked with a non-preprocessed script, Arlopp is invoked to create a temporary script object. The normal practice when developing an application is to work with an Arlo source script and use the built-in script editing facilities to modify and enhance the program. When this process is finished, Arlopp is used to create a permanent object file for the installed application. [5]

5. Arlo Source

Arlo source files consist of items in TIPs [6] data base format. These items consist of records and fields that: specify the item name and its aliases, the item title, its initial status, a monitoring name, an explanation and explanation command, the names of the tableau (menu) and enable groups to which the item belongs, other script files the item requires to be included, and, most importantly, the Arlo commands.

An annotated example of a source item is given in a later section.

6. Arlochk

The major objective of Arlopp is to convert the Arlo source to relocatable units and to build the item access information. As such it does little checking of the syntax of the individual commands and there are many errors that may occur during interpretation. As a programmer aid, Arlochk processes an Arlo object file looking for as many run-time errors as it can detect. It reports syntax errors in expressions, missing files, invalid variable uses, unreachable commands, unused items or variables, missing keyword command arguments, and so on.

7. Arlohdr

The Arlohdr program is another programmer aid. It examines an Arlo script object and produces various statistics and tables as specified by option flags.

8. Arlolog

Arlo supports a form of command usage monitoring that will record any invocation of an item that has a 'monitoring string'. The output file contains the user name, the time the command was invoked, and the monitoring string, all in a binary record. Arlolog 'unpacks' this record and prints

[5] Script preprocessing can be expensive, so using an already preprocessed script is time-saving. However, many users don't bother and just accept the expense of preprocessing every time a script is used.

[6] TIPs is a text structuring system. TIPs items consist of a 'name' record and tagged data records. The data records (and the name record) can be either free form text, tab or newline separated lists, or field records. A fuller description of TIPs may be found in Introtips(1).

out the log in a readable form.

9. The Arlo Host File system interface

Arlo provides an interface to the Host file system. The script may read from or write to files. It may test a file's attributes and change the location of the current working directory.

10. Arlo Host command system interface

Arlo can invoke commands via the Host command system. In fact, this could be viewed as its major function. On UNIX systems, it uses the shell to invoke programs and can arrange to read or write the program's standard input and output.

11. Arlo Msg interface

Some applications use Arlo as a front end to an application server. For example, Arlo is being used within IST to interface to forms editors and binary relational data bases.

Currently the message interface is constructed using multi-plexed files on 4.1BSD. However, due to the lack of a consistent IPC mechanism across all UNIX systems, this is being re-implemented using pipes.

Basically, the script may invoke another program that is attached to the original application via some form of communication channel.

12. Xarlo

Arlo has over 85 different keywords, 160 error messages, 30 control registers, 15 special read-only registers, and 9 special items. This requires a great deal of documentation. As a programmer aid, descriptions of all aspects of Arlo are contained data bases. These can be accessed via programs collectively called 'Xarlo' [7] which provide keyed access to the documentation. For example, the short description and synopsis of the 'replace' keyword can be obtained using the command 'xarlo replace'. A full description may be obtained using 'xarlok replace'.

An Annotated Arlo Item

This section presents and describes a contrived example item to explain Arlo items and the Arlo programming language.

TIPs format uses tabs and newlines to separate fields. To ensure that the tabs are visible, they are replaced by ' Θ ' in the following. The words in **bold** are Arlo keywords.

```
*na Θ now Θ What time is it Θ on Θ lognow
*al Θ time Θ date
*ta Θ adm Θ main
*en Θ adm
*sy Θ now [ timezone ]
*sy Θ time [ timezone ]
*ex Θ This item displays the time and date.
If invoked as 'time', only the time is output.
If an argument is given, it is taken to be a timezone,
and the time is relative to that zone.
*ex Θ Known time zones are:
*ec Θ sh Θ rotf --01 zones
*no Θ This is an example item
```

---

[7] Xarlo is constructed using Mkhlp(1), the source for which is created by processing the data bases which are used to define the various aspects of the Arlo system. All Arlo error messages, keywords, and special variables are described in other TIPs data bases which are then processed to create the required C tables and Mkhlp input.

```
*co ⊕ / SccsId %W% - %E%
*co ⊕ strcmp ⊕ `($0) ⊕ time
*co ⊕ [_T= =0] assign ⊕ format ⊕ The time is %r
*co ⊕ [else] assign ⊕ format ⊕ %r, %h %d 19%y
*co ⊕ ($1) assign ⊕ tz ⊕ `($1)
*co ⊕ [else] askval ⊕ tz ⊕ In what time zone
gmt ⊕ Enter the time zone for which you want the time
in lower case letters.
*co ⊕ (tz)
*co ⊕ - getline ⊕ #o ⊕ !grep `(tz)' zones | rotf -2
*co ⊕ - [_S] enditem ⊕ Sorry: zone `(tz)' unknown.
Use '?now' for a list of known zones.
*co ⊕ [else][o=0]
*co ⊕ date ⊕ output ⊕ _B+o*3600 ⊕ `(format)
*co ⊕ call ⊕ title ⊕ `(output)
*us ⊕ title.us
```

The following is a record by record description of the example item.

```
*na ⊕ now ⊕ What time is it ⊕ on ⊕ lognow
```

The name record is the only required record of an Arlo item in that it begins the new item and terminates the previous one. The name record consists of four fields, as follows:

name ('now' in the above record) This field is the primary name of the item. The user or another item may invoke the item using this name or one of the aliases. Arlo has a set of special item names that are used in special circumstances. For example, the 'badchoice' item is used whenever the user's response to the selection prompt is not the name of a valid and enabled item. If the 'name' field is the name of a special item, the item is used for that special function [8].

The name field is the only one that is required.

title ('What time is it' in the above record) The title field should be a short description of the item, and where necessary should indicate the optional arguments. The title field is displayed in Arlo 'Tableaus' ('menus').

status ('on' in the above record) The status field must be either 'on' or 'off', where 'on' is the default. If an item's status is 'off', it may not be selected by the user. This field is normally used to disable an item that is used as a subroutine by other items, or to disable an item until some condition is met. There are keywords that change the status of items.

logstr ('lognow' in the above record) The Arlo system provides a mechanism whereby item usage can be logged in a file (see 'Arlolog'). If item logging is turned on and an item that has a non-null logstring is invoked, the logstring, the time and the user's userid are appended to the logfile.

```
*al ⊕ time ⊕ date
```

The alias records are alternative names for the item. The primary name is specified in the first field of the name (na) record. When an item is invoked the name or alias used is available as the first argument. If the example item is invoked using the alias 'time' just the time is output, whereas if invoked using either 'date' or 'now' the date and time is output.

```
*ta ⊕ adm ⊕ main
```

[8] The assignment of special items can be changed dynamically within the script.

Frequently Arlo scripts contain far too many items to comfortably present in a single tableau. Studies have indicated that 8 items in a menu is about optimal. The 'tableau group' record is used to divide a script's items into groups that can be individually displayed in a tableau. The example item belongs to the groups 'adm' and 'main'. Special names can be used to assign an item to all groups ('*') or to exclude it from all ('-') or named groups (e.g., '!main').

```
*en ⊕ adm
```

Often an interactive application will have commands that should not be enabled until some action has been performed or some condition is satisfied. The 'enable' group records are used to assign the item to an enable group, which can then be enabled (disabled) using the 'grpon' ('grpoff') keyword.

```
*sy ⊕ now [ timezone ]
*sy ⊕ time [ timezone ]
*ex ⊕ This item displays the time and date.
If invoked as 'time', only the time is output.
If an argument is given, it is taken to be a timezone,
and the time is relative to that zone.
*ex ⊕ Known time zones are:
*ec ⊕ sh ⊕ rotf --01 zones
```

One of the Arlo design objectives was to make it as easy as possible for the programmer to enter documentation and to make it trivial for the user to obtain that documentation quickly. The 'sy', 'ex', and 'ec' records (Synopsis, Explanation, and 'Explanation Command') are used to contain the documentation for an item. If the user responds to the selection prompt with a question mark ('?') followed by the name or alias of an item, the item's synopsis and explanation records are output and its explanation command records are interpreted. In the above example, the '*ec' record will cause the time zone abbreviations and names to be output using a shell command.

```
*no ⊕ This is an example item
```

The notes records serve no purpose other than to act as comments for the programmer and those that follow him/her. Comments can also be included in the command records. However, such comments are included in the output object file and, therefore, add to the cost of interpretation.

```
*co ⊕ / SccsId %W% - %E%
```

The '*co' records contain the commands that are interpreted when the item is invoked. The command records usually contain two fields: the conditional guards and keyword, and the keyword's arguments. The above example is a comment, which is indicated by using a '/' instead of a keyword. Such a comment should be used instead of 'notes' records when it is important or convenient to have the comment embedded in the object code. Arlo provides facilities whereby the user can examine the code in the object file during interpretation.

```
*co ⊕ strcmp ⊕ `($0) ⊕ time
*co ⊕ [_T= =0] assign ⊕ format ⊕ The time is %r
*co ⊕ [else] assign ⊕ format ⊕ %r, %h %d 19%y
```

In the first command, **strcmp** is the keyword, and '`($0)' and 'time' are the keyword arguments. The '`($0)' is similar to '$0' in the UNIX shell. Anytime the value of a string variable is used in Arlo, the variable name is enclosed in parentheses after a '`'. The '[_T= =0]' and '[else]' on the second and third lines are guards. The **strcmp** keyword sets the '_T' register to 0 if the two argument strings are equal. The guard '[_T= =0]' is evaluated on the next line and, if it succeeds, the keyword **assign** is interpreted and the string 'The time...' is assigned to the variable 'format'. If the guard fails, then the keyword is not interpreted. In the above, if the guard on the second line fails, then the third line will be interpreted [9].

[9] The '[else]' guard is actually testing the value of the control variable 'else'. The value of 'else' is always the ne-

```
*co ⊕ ($1) assign ⊕ tz ⊕ \($1)
```

In the above statement, the guard is '($1)'. Such guards succeed if the string named inside the parentheses (i.e., '$1') is non-null. If the string '$1' is null then the keyword and any indented statements that follow are skipped. If it isn't null, then the keyword is interpreted. In this statement if '$1' is not null, then its value is assigned to the variable 'tz'.

```
*co ⊕ [else] askval ⊕ tz ⊕ In what time zone
gmt ⊕ Enter the time zone for which you want the time
in lower case letters.
```

The above statement is guarded by '[else]'. Therefore it will be interpreted, only if the guard on the previous statement failed. This statement asks the user for a response to the question 'In what time zone'. The **askval** keyword takes 4 arguments. The first ('tz' in our example) is the variable to which the user's response is assigned, the second is the question posed to the user, the third is the value used if the user responds with an empty line (i.e., the default value), and the last is the explanation given the user if he/she responds with a '?'.

```
*co ⊕ (tz)
*co ⊕ - getline ⊕ #o ⊕ !grep '\(tz)' zones | rotf -2
*co ⊕ - [_S] enditem ⊕ Sorry: zone '\(tz)' unknown.
Use '?now' for a list of known zones.
*co ⊕ [else][o=0]
```

The first statement is a guard of the two indented statements that follow. Note that Arlo uses indentation (indicated by minus signs) to create statement blocks, whereas languages such as C or PASCAL use block delimiters (e.g., '{...}' or 'BEGIN...END'). In the above, the indented block will be interpreted if the string variable 'tz' is non-null, otherwise the block is skipped [10]. The second statement 'grep's for the 'tz' value in the 'zones' file and selects the third field of the first found line and assigns the result to the number variable '#o'. Any use of shell commands cause the '_S' register to be set to the exit status of the shell command, which in this case would be the exit status of the 'grep'. The guard on the statement that follows the 'grep' will succeed if 'grep' failed. The actual command causes the item to exit with a message to the user stating that the timezone is unknown and that a list of timezones is available via the explanation. The last statement above illustrates the use of multiple guards. The second one (i.e., '[o=0]) is evaluated only if the '[else]' succeeds. The effect of this statement is to set the number variable 'o' to zero, if a timezone wasn't specified.

```
*co ⊕ date ⊕ output ⊕ _B+o*3600 ⊕ \(format)
*co ⊕ call ⊕ title ⊕ \(output)
```

The command **date** is used to format the current time (the register '_B') plus the timezone offset (o*3600). The result of formatting the expression is assigned to the variable 'output' which is then passed as an argument to the item 'title' which will display it.

```
*us ⊕ title.us
```

The 'title' item is not actually contained in this script, but in another file called 'title.us'. The '*us' (use file) record causes the named file to be included in the current script. The user can specify a path of directories to be searched for the 'use file'.

gation of the result of the evaluation of any guards on the previous statement.

[10] The guard will actually never fail because the previous lines ensure that 'tz' has a value since the user is asked for the timezone if not already set and a default is provided. Therefore the test of 'tz' is redundant. It is done this way to illustrate Arlo structures, not efficient coding practices.

The Example Item in Use

The following is an example user/arlo dialogue which exercises the example item described in the previous section. The user's responses are in **bold**.

```
Selection prompt] now
In what time zone: ?
Enter the time zone for which you want the time
in lower case letters.
In what time zone: bst
Sorry: zone 'bst' unknown. Use '?now' for a list
of known zones.

Selection prompt] ?now
now: What time is it

Aliases: time, date

Usage:  now [ timezone ]
        time [ timezone ]

This item displays the time and date. If invoked
as 'time', only the time is output. If an argument
is given, it is taken to be a timezone, and the
time is relative to that zone.

Known time zones are:

        gmt     Greenwich
        est     Eastern Standard Time
        pst     Pacific Standard Time
        nsw     New South Wales Time

Selection prompt] now gmt

03:17:11 PM, Oct 12 1984

Selection prompt]
```

Building an Arlo Application

Within IST, there are a number of methods used to create Arlo scripts. For example, the IPSE project team has developed and is using an Arlo work-bench system (programmed in Arlo) that helps the user build, maintain and test Arlo scripts.

However, the most common way (and the easiest) is to perform the following shell commands:

```
echo "*na ⊖ startup" >arlofile.as
arlo
```

The above creates an Arlo script and then starts up the Arlo interpreter for that script [11].

[11] Like the UNIX utility *Make* (1), Arlo allows the script file to be either specified by argument, or to default to a file in the current directory. The file 'arlofile.as' is one of the file names it will accept. The '.as' suffix is not required, but is a convention that we have adopted to indicate Arlo source files.

At this point, the application does little more than prompt the user for a command. However, one of the commands that it will accept is 'E' which will invoke the user's editor to edit the current script, and upon return will reload that script. Thus the script can evolve.

More important than the mechanics of script evolution is the way in which a script is designed and implemented. Perhaps one of the most powerful features of Arlo is the ability to construct scripts in a top-down manner.

The first step should be for the designer to create the name and explanation records [12] for the application's primary items, and, if necessary, the tableau controls (i.e., the divisions of items into tableau groups and the mechanisms that change from one tableau to another).

This should be immediately tested!

The next step is to fill in some of the commands for those items, and then test those commands. Where the item being tested requires subroutines or facilities that are not available, the programmer should use the Arlo debugging keywords to perform the required processing manually.

The above steps are repeated until the application is ready for publication. The script is then, if necessary, processed by Arlopp to produce a pre-processed Arlo binary which is installed. But note, that when using the installed application, facilities exist to make and test modifications (provided file permissions permit it) easily and quickly, without leaving the application.

Conclusions

Arlo has proved to be a very useful tool for both long and short term applications. Furthermore, it is often the case that short term applications evolve into installed applications [13].

Applications are relatively easy to build, and when built are self-documenting. Consistency of user interface is preserved, both within a single application and across a range of applications.

Arlo was designed to meet a perceived need for a 'system integration' mechanism - the 'glue' that would bind a system together, addressing both component integration and user interfacing. User experience to date indicates that the perceived need was a very real one, and that the Arlo system is effective in addressing that need.

[12] At one site, the primary user application contained an item that allowed the user to specify a new item to be added to his/her application. The user was prompted for the name, title, and explanation which was then mailed to the support group. The support group, if possible, added the necessary code and delivered the completed item to the user's script. The next time the user loaded the script, the item would be incorporated.

[13] Our mail system started life as a very simple script that split incoming mail into separate files and then helped the user to browse the directory data base using the standard UNIX tools. It has now become a full scale mail system with forwarding, addresses aliases, and multiple mail folders, yet it only uses standard general purpose tools, a couple of TIPs utilities, and a program that maintains and outputs a unique letter identifier.

NAME

introtips introduction to TIPs

DESCRIPTION

TIPs (the Tilbrook Information Processing System) is a data base management and output system for small text data bases. It was designed and created by David M. Tilbrook while he was at Human Computing Resources for Soft Arkiv (both in Toronto, Canada) as a demonstration at the 10th annual Sculpture Conference, which was held in June 1978 at York University in Toronto.

Further extensions have been made at Bell-Northern Software Research, Systems Designers Ltd., and Imperial Software Technology including the development of a library of subroutines that facilitate creation of special applications.

This manual section will briefly describe TIPs data bases and software. More descriptions are available in the referenced manual sections.

THE DATA BASES

It should be noted that all the following relates to the general purpose TIPs tools and concepts. Some of the descriptions may not be applicable or valid when considering special purpose TIPs systems (i.e., *Arlo*). A TIPs data base consists of a descriptive file (henceforth called the profile) and one or more data files. Normally, a data base is contained in a directory of its own. There will be two files called *profile.tp* and *.Tprof*. These are the original initialization and the binary profile respectively. The data files should be suffixed by '*.d' if they are to be read as the default files.

The profile contains the names and structures of all classes of information in the data base. For a full description of profiles and how they are constructed, see *Tmkprof*(1T).

Note that shared or frequently used profiles are usually stored in '/usr/ist/lib/tips', and that all the TIPs programs that require profiles support a '-p profile' that will access profiles stored in this directory.

The data files are normal UNIX text files (thus they can be edited and created with standard UNIX tools). They can contain zero or more complete entries.

An entry can be thought of as a set of records describing a single object. For example, in a data base of sculptors, an entry would be the bibliographic information about one artist.

All entries are divided into classes (i.e., records) of information. Each record contains information about one aspect of the entry. For example, in the artist data base, there could be records for the artist's birth date, address, or exhibitions. Each record has a unique two letter identifier (henceforth called the tag).

For the most part, an entry can contain zero or more instances of any record. There is no required order for the records, with the exception of the 'name' record (the tag is always 'na'). Every entry must begin with an 'na' record, and the occurrence of a subsequent 'na' line indicates the beginning of the next entry.

Records are entered as lines of text. The first four characters of the first record line are an '*', the tag, and a tab character (ASCII 011). Such lines will hence forth be called 'tag lines'. Normally the text for a record ends at the next tag line or EOF (i.e., End of File). The exception is the RAWDATA type record which ends at the next 'na' tag line or EOF.

There are five basic classes of records. These are: 1) FIELDS; 2) LIST; 3) TEXT; 4) PHRASE; and 5) RAWDATA. The record's type is set in the profile.

FIELDS type records are used to contain information that consists of related parts. For example, in the artist data base, the artist's address was contained in a FIELDS type record. The fields were street, city, province or state, country and postal code. In the data files, fields are separated from each other by tab characters, or NEWLINES (ASCII 012). The last field of a FIELDS type record can be narrative text. It may contain literal tab and newline characters. (See *Tmkprof*(1T) for complete description of the text entry.)

LIST records are used to contain groups of equivalent units of information (e.g., the list of organizations to which an artist belongs.) In the data files, a LIST record line can contain as many list elements, separated by tab characters, as desired.

TEXT records are used to contain text. The text begins immediately after the tab following the tag, and continues up to the next tag. Any contained tab characters are kept.

A PHRASE type record is a special case of a TEXT record. The only difference is that there is a maximum of 1 field per record.

EXAMPLE DATA BASE DIRECTORY

The following could be a list of files in a sample TIPs data base.

.Tprof	Binary profile
data01.d	a data file
data02.d	YADF (Yet another data file)
profile.tp	Profile initialization

The 'profile.tp' file is required by *Tmkprof*(1T) only.

The files 'data*.d' are the files containing the TIPs entries. An example entry follows. For readability, tab characters are shown as ' <> '. Each record in the entry is numbered for reference. (The example is from the data dictionary of the UNIX/PWB kernel).

```
1) *na <> bmap <> os/subr.c <> 16
2) *us <> os/rdwri.c <> 46 <> readi
  if ((bn = bmap(ip, lbn)) == 0)
3) *us <> os/rdwri.c <> 98
  writei <> if ((bn = bmap(ip, bn)) == 0)
4) *ky <> filesys <> block
5) *un <> 18-6 <> 6415
6) *sy <> bmap(ip,bn)
7) *de <> Bmap defines the structure of file
  system storage by returning the physical block
  number on a device given the inode and the
  logical block number in a file.
  When convenient, it also leaves the physical
  block number of the next block of the
  file in rablock for use in read-ahead.
8) *ky <> extra
```

The 'na' record (#1) is the first record of the entry. The entry continues up to but not including the next 'na' record, or an End of file.

The 'na' record and the 'us' and 'un' records (#2, 3, 5) are all FIELDS type records. Each field in the record holds a different piece of information about the item referred to in the record. For example, in the 'us' (Usage) records the contained fields are: the source file name, the source line number, the containing function, and the contents of the line, in that order. Record #3 actually consists of two text lines. The NEWLINE following the source line number field is interpreted as a TAB since the next line is not the start of a new record. It should be noted that there are two 'us' records. All records (with the exception of an 'na') can be repeated as many times as is required.

Records #4 and #8 are both 'ky' (keyword) records. The 'ky' record is a LIST type record. Each field of all the 'ky' records are considered as elements in the 'ky' list. LIST record elements can be entered as individual records or as part of the same record with separating tabs.

The 'sy' and 'de' records (#6 and #7) are TEXT type records. In such records, the content of the record is the text stream up to but not including the next tagged line. For example, record #7 is terminated by the occurrence of record #8. The tabs and newlines in the text are considered part of the text.

THE SOFTWARE

TIPs consists of a variety of major general purpose programs or subroutines. These are *Tscan*, *Tlist*, *Tprof*, *Trgmk*, *Trg*, *Trgdmp*, *Tmkprof*, *Tdbm*, *Tdbrg*, *Tdbkeys*, and the TIPs subroutine library, */usr/lib/libtips.a*.

Tscan allows a user to interactively scan a data base. It allows entry selection by a variety of techniques and display of the text under selective control. For example, it is possible to scroll through the data base, selecting entries by query, index, or name, one by one, or a screenfull at a time, and then ask for records or fields to be listed automatically or by command. See *Tscan*(1T).

Tlist allows the user to list a data base's contents in a variety of formats. An option allows listing of only entries that match a specified query. Another option checks the data base files for syntax errors. The formats offered by this program are fairly weak. The use of *Trg* (see below) will allow the user more power and flexibility in structuring the output. *Tlist* has two useful options which are: '-c' -- copy selected entries to standard output in TIPs format; and '-e' -- check specified files for TIPs format errors. See *Tlist*(1T).

Tmkprof is the process used to create a new profile for a data base. The profile is a set of tables that is loaded into core (see *getprofile*(3T)) prior to reading any TIPs data base. *Tmkprof* parses a description of the records and fields of a data base and creates the binary form of the profile, usually in a file called ".Tprof".

Tprof will produce, on standard output, information about the profile. This information is available in a variety of formats. One will create the actual input to Tmkprof that created the profile. Another will produce a C source file that can be used to create static profiles for use in special purpose applications.

Trg, *Trgmk*, and *Trgdmp* are tools used to convert TIPs data base entries into other data formats. The user prepares a template which is compiled and then interpreted for each selected entry in the data base using *Trgmk*. The template can contain strings, tag names, output controls and flow constructs. *Trg* interprets the object file for each selected data base entry. *Trgdmp* is a tool that will dump the object file in a readable form. One of the options is to convert the object file to C code which can then be compiled as part of a program, a feature that is used in some applications. *Trg* is useful for creating other data bases, or preparing data for processing by other UNIX tools (e.g., *Nroff*(I)).

Ted is a TIPs file editor. It allows the user to move and create new entries and to edit individual records and/or fields. At IST, there is a screen editor that understands TIPs profiles and format that may be invoked from Ted.

Tdbm allows the creation of a key-search index front-end for TIPs database files to cut down on disc accesses for large databases.

Tdbkeys shows the keys created by *Tdbm.*

Tdbrg is a variant of *Trg* that uses *Tdbm* index files to select and output entries using a *Trg* program.

/usr/lib/libtips.a is the library that contains all the standard TIPs routines. For example, the routines to do TIPs entry I/O, parse and evaluate queries, load profiles, retrieve field or records instances are all contained in this library. See *Xltips*(1T).

Xltips(1t) is a *Mkhlp*(1) facility that provides online documentation about the TIPs library. The are a number of other *Mkhlp* facilities that explain other aspects of TIPs and the supporting tools:

xtips	general TIPs information
xltips	the TIPs library routines
xtrg	the builtin functions of *Trg*
xtscan	the prompts of *Tscan*
xted	Error messages and commands in *Ted*

These TIPs libraries have been used extensively to implement special purpose data base systems such as *Contax*(1); *Alice*(1); (the "you kin git anythin y'want at Alice's restaurant" menu system.) *Arlo*(1) successor to Alice; *Hlp*(1) the software inventory; *Ma*(1) the mail system; *Termcap*(1) a real termcap data base; *Site*(1) UUCP site map.

SEE ALSO

tscan(1t), tlist(1t), tmkprof(1t), tprof(1t), tdbm(1t), tdbkeys(1t), tdbrg(1t), trg(1t), trgdmp, trgmk(1t), and The TIPs User Tutorial

FILES

/usr/ist/lib/tips/* -- public profiles
/usr/ist/lib/tips/forms/* -- public editor forms
name/.Tprof (name is DB's directory name) - source and binary of profile.
name/*.d (name is DB's directory name) - data files for DB name.
/usr/lib/libtips.a -- archive of TIPs building blocks

AUTHOR

David Tilbrook

BUGS

In a couple of places, it does not use Stdio routines due to efficiency problems.

EUUG NATIONAL GROUPS

UKUUG

Chairman — Sunil Das,
Computer Science Dept.
The City University,
Northampton Square,
London, EC1.
England

Tel: 01-253-4399

NLUUG

Chairman — Teus Hagen,
ACE,
Associated Computer
Experts Bv,
nz voorburgwal 314,
1012 RV Amsterdam,
The Netherlands.

Tel: 020 24 54 44

Secretary — Dr. M. van Gelderen,
NIKHEF-K,
Kruislaan 411,
Postbus 4395,
1009 AJ Amsterdam,
The Netherlands.

EUUG-S

Chairman — Bjorn Eriksen,
ENEA Svenska AB,
Box 232,
S-183 83 Taby,
Sweden.

Tel: 46 8 756 72 20

Secretary — Hans Johansson,
Logica Svenska AB,
Norra Stationsg. 79-81,
S-113 33 Stockholm,
Sweden.

Tel: 46 8 34 91 10

AFUU

Chairman — Mr. Bernard,
c/o 152 bis,
avenue Marx Dormoy,
92120 Montrouge,
France.

Tel: Montrouge 655 45 50

Secretary — Ms. J-M Langlade,
c/o 152 bis,
avenue Marx Dormoy,
92120 Montrouge,
France.

Tel: Mntrg. 655 45 50

DKUUG

Chairman — Keld-Jorn Simonsen,
Indre By-Terminalen,
University of Copenhagen,
Studiestraede 6 o.g.
DK-1455 Copenhagen K,
Denmark.

Tel: 45 1 1201 15

FUUG

Chairman — Heikki Arppe,
Technical Research Centre
of Finland,
VTT/ATK,
Lehtisaarentie 2A,
2F-00340 Helsinki,
Finland.

Tel: 358 0 4561

Vice
Chairman — Johan Helsingius,
Tontunmaentie 32 AS,
02200 Espoo 20,
Finland.

Tel: 358 0 460 033

GUUG

Chairman — Johannes Koehler,
GUUG,
Hauptstra 60,
8520 Erlangen,
Germany

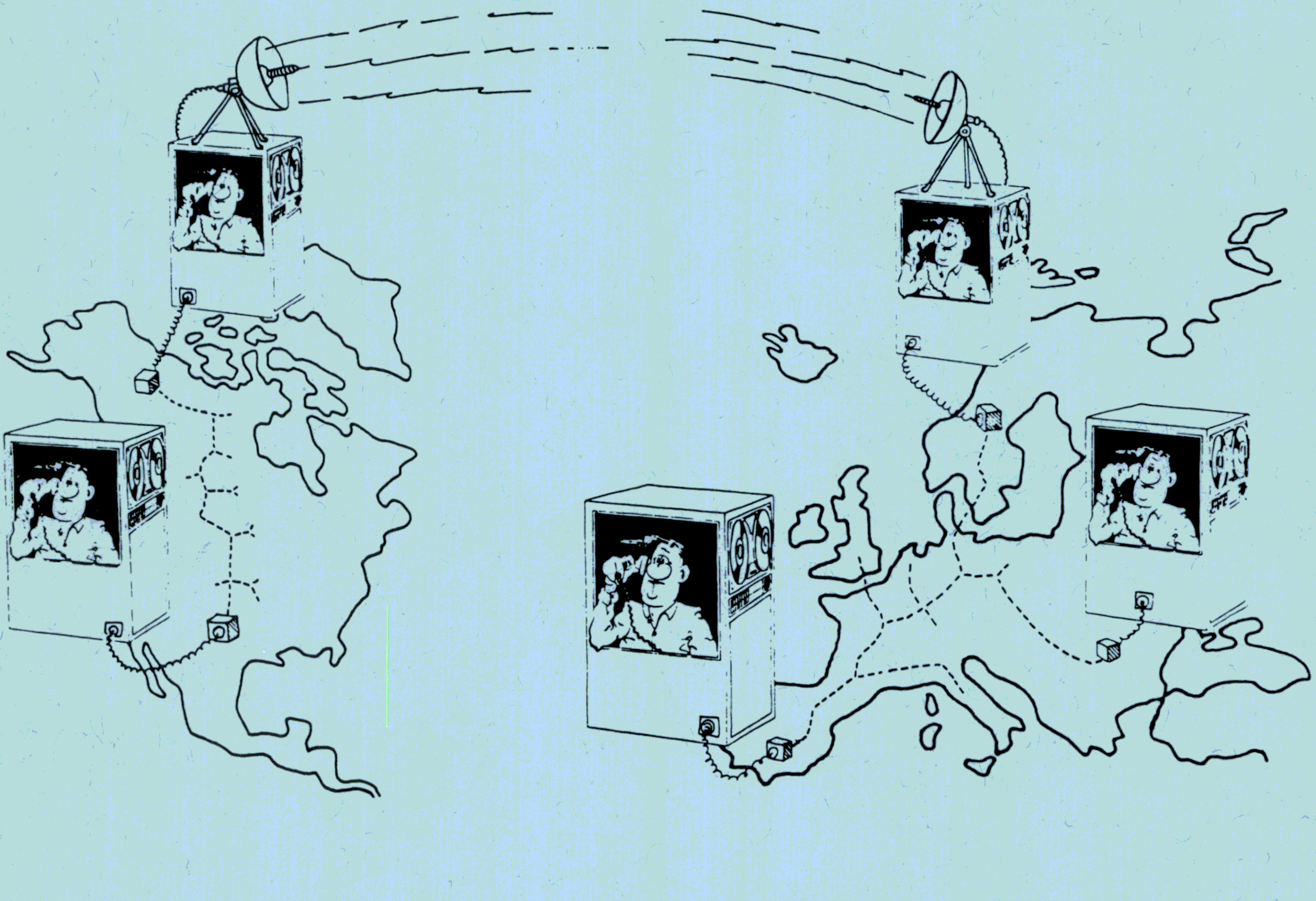
Vice-
Chairman — Daniel Karrenberg,
Universitaet
Dortmund,
Informatik IRB,
Postfach 500500,
D-4600 Dortmund 50
Germany.

Tel: 231 755 2444

EUUG-I

Chairman — Giuseppe Molinari,
Systems & Management spa,
Piazza Solferino 7,
10121 Torino,
Italy.

Vice-
Chairman —Marco Mercinelli,
CSELT,
10148 Torino,
Via G. Reiss
Romoli 274,
Italy.



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