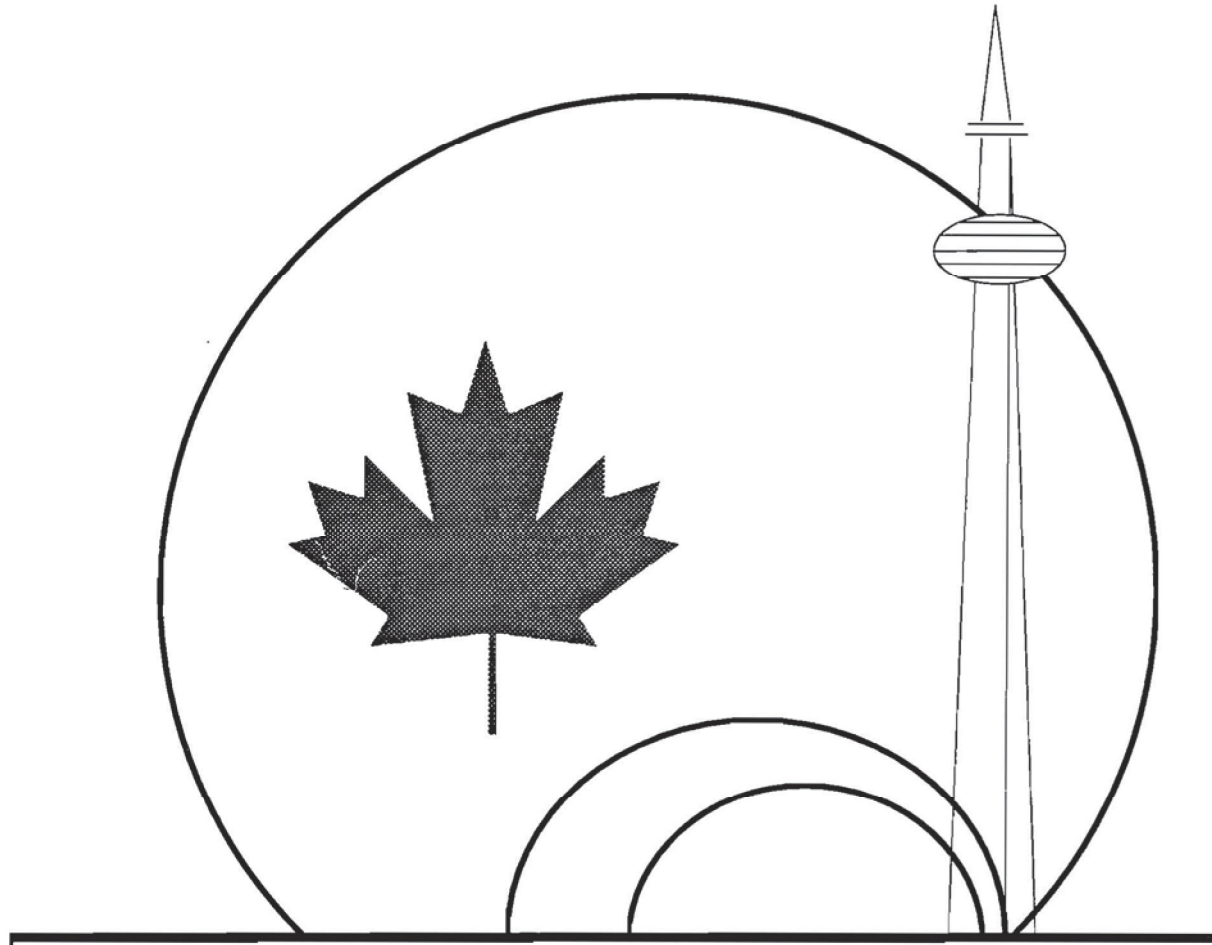


# EXHIBIT D

1996  
ROCHESTER  
FORTH  
CONFERENCE

# OPEN SYSTEMS



© N. Solnsteff

June 19-22, 1996  
RYERSON POLYTECHNIC UNIVERSITY  
Toronto, Ontario  
Canada

Copyright 1997  
by the Institute for Applied Forth Research, Inc.

Individual papers in this volume maybe  
copyrighted by the author(s) or company  
responsible for the work. The Institute  
makes no claims for the correctness of the  
material presented.

**ISBN# 0-914593-16-1**

A publication of

The Institute for Applied Forth Research, Inc.  
70 Elmwood Avenue  
Rochester, NY 14611

Published in the United States of America  
Printed in the United States of America

## TABLE OF CONTENTS

Starting Kiln 4 with Forth Inc's EXPRESS	<i>Allen Anway</i>	1
Open Systems - Theory and Reality	<i>Steve Benson</i>	3
Using Forth as a Scripting Language	<i>Dr. Everett F. Carter Jr.</i>	9
This Android Goes Forth Openly and Objectively	<i>Paul Frenger M.D.</i>	13
The GENETIX Software Genomes	<i>Bernard A. Hodson</i>	19
The Europay Open Terminal Architecture: A Forth-based Token System for Payment Terminals	<i>Peter Johannes, Elizabeth D. Rather and Stephen Pelc</i>	27
Microcontroller Forth: A Windows-Based Development Environment	<i>Viacheslav Kirillin, Andrey Kloubovitch and Dmitri Preobrazhensky</i>	31
Microcontroller Forth: An Approach to Hardware Programming	<i>Oleg Klementiev and Dmitri Preobrazhensky</i>	43
The Evolution of Development Tools for the Forth Language	<i>Randy Leberknight</i>	47
Booting Martian Software	<i>Benjamin Lee</i>	53
Proposed ANS-Forth C-Interface Word Set	<i>Marty McGowan</i>	57
The Evolution of SENDIT into EPIC	<i>Stephen Pelc</i>	63
Highlighting a Scheme for Threaded-Code Optimizing*	<i>Ewald Pfau</i>	69
A Concept for Medium Sized Relational Database Programming*	<i>Ewald Pfau</i>	71
The Europay Open Terminal Architecture: A Forth-based Token System for Payment Terminals	<i>E. D. Rather, Peter Johannes and Stephen Pelc</i>	73
A Small 32-bit RISC Core	<i>John Rible</i>	79
A System for Distributed Inferencing*	<i>Bradford J. Rodriguez and W. F. S. Poehlman</i>	85
Martian Memory Management	<i>Ray St-Laurent and Benjamin Lee</i>	87
Standard Tools that are Standard Code themselves	<i>J E Thomas</i>	91
Laws of Computing, Version 2.0	<i>C. H. Ting</i>	93
Traffic Controller and Zen of State Machines	<i>C. H. Ting</i>	103
The Sensor Driven Airborner Replanner Simulation: Building The World in a Box	<i>Richard C. Wagner</i>	109

---

\* Abstracts only are included for these presentations.





## Introduction

The 1996 Rochester Forth Conference on Open Systems was held at Ryerson Polytechnic University, in Toronto, Ontario, in Canada June 19-22, 1996. This was the first Rochester Conference to be held outside the United States, and was a great success. Attendees came from Canada, the United States, England, the Netherlands, Australia, Austria, and Russia.

The Conference was hosted by the Institute for Applied Forth Research, Inc., in cooperation with the Southern Ontario chapter of the FORTH Interest Group. The Conference was sponsored by New Micros, Inc., with the continuing support of Mr. Randy Dumse; ITV Corporation, with the new support of Mr. Joe Zott; Ross Technology, Inc. and the support of Mr. Warren Bean and JWK International Corporation, with the continuing support of Dr. Jay Khim. Additional support was provided by Xela Associates, and Alexander Forsley, of Annandale, Virginia and Miller Microcomputer Systems, of Natick, MA.

During most Conferences the apparent theme appears, in contrast to the advertised theme. This year was an exception. Several papers dealt with different aspects of Open Systems. These include an overall view of Open Systems by Dr. Steve Benson, Forth as a scripting language by Dr. Everett Carter; and GENETIX by Bernard Hodson. An even larger view of Open Systems was described in a paper by Peter Johannes,

Elizabeth Rather and Stephen Pelc regarding Europay and a background paper on the supporting software, EPIC, by Stephen Pelc. These efforts are all the more far reaching by supporting banking institutions in a variety of countries on a variety of hardware platforms. Finally, two remarkable papers: John Rible presented a small 32-bit RISC Core and Richard Wagner gave details of an airborne simulation.

The Conference Program Chair and Proceedings Editor was Dr. Nicholas Solnsteff and the Facility Chair was Dr. Bradford Rodriguez. I was delighted to have their strong support. In large measure they organized and orchestrated this Conference, resulting in a great success! My thanks goes to both of them, and I look forward to working with them on future Conferences.

No conference can be a success without a large supporting cast. We had very able assistance from: Elliott Chapin, Kirby Dumse, Alexander Forsley, Brenda Forsley, Beverly Galloway, Ken Kupisz, Ken McCracken, Rob McDonald, Wendy Rodriguez, Jon Verne and Robin Ziolkowski. I wish to complement and thank the very helpful Ryerson University Staff, most notably Patti Franklin, of the International Living Center and Peter Young, of the International Conference Center.

--Lawrence P. G. Forsley  
Conference Chairman

## The Europay Open Terminal Architecture A Forth-based Token System for Payment Terminals

*E. D. Rather,*  
*FORTH, Inc.*  
*111 N. Sepulveda Blvd., Suite 300*  
*Manhattan Beach, CA 90266 USA*

*Peter Johannes,*  
*Europay International*  
*Chausée de Tervuren 198A*  
*B-1410 Waterloo, Belgium*

*Stephen Pelc,*  
*MicroProcessor Engineering, Ltd.*  
*133 Hill Lane*  
*Southampton, England*

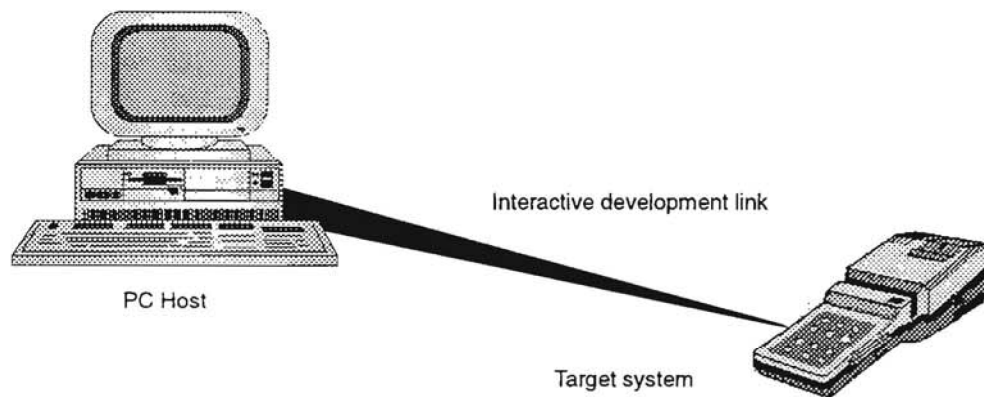
### Abstract

Europay, the major European credit card organization (Eurocard, MasterCard in Europe, and other financial systems) is developing technology to support "smart" cards — Integrated Circuit Cards, or ICCs — in the credit cards of the future. This will require new software in all credit card terminals (ranging from 8051-based POS terminals to high-end ATMs). To facilitate this transition, they are designing a token-based system conceptually similar to Open Firmware or Java based on Forth. Using the "Open Terminal Architecture" (OTA) it will be possible for credit card issuers and acquirers to write application programs that will be completely platform independent, and run on all OTA-compliant kernels.

This is the second of three papers on OTA; it will discuss the overall architecture of OTA.

### System Components

The purpose of an OTA system is to provide software to run in terminals used in payment applications. Conceptually, there are two hardware environments, and several classes of software. The hardware environments include the *development system*, which is based on a simple PC, and a *target* which is some form of payment terminal. The software includes *development software*, which runs on the PC; *kernels*, which include all platform-specific software in a terminal and other mandatory standard functions; *libraries*, which provide general functions to support terminal programs and payment applications; *applications*, which are the functions specific to a particular payment product, and *terminal programs*, which perform general non-payment terminal functions and include high-level mechanisms for selecting and executing transactions and associated applications.



**Figure 1**

An OTA development environment for a terminal small program to support the terminal end of the Interactive Development Link (IDL) protocol is usually placed in the terminal RAM.

### Development Environment

An OTA development system is used to develop terminal software, either low-level kernel software or high-level library or application software. Kernel development requires a target terminal to be connected, as the kernel is cross-compiled on the PC host and downloaded to the terminal across the Interactive Development Link. OTA libraries, terminal programs, and applications are also developed on a PC host. Because they are high-level code, they may also be executed on the host for preliminary testing, using the host's version of the standard kernel, and



thus a target is not needed. For final testing using the terminal's own kernel and I/O, these programs would be tokenized (see discussion below) and downloaded to the target terminal.

As an example, a prototype development system consists of a PC running Windows 3.11, Windows-NT or Windows 95 and an IDL to a target terminal (see Figure 1). Development software on the PC is based on ProForth for Windows, a product of MicroProcessor Engineering Ltd.

### **Terminal Target Environments**

The target system is any one of a large variety of payment terminals. Actual products range from small, hand-held devices with simple 8-bit microprocessors such as the 8031/51 family to 32-bit computers running operating systems such as Windows-NT or Unix. In order to simplify the production, certification and maintenance of software on such a wide variety of targets, OTA terminal code is based on a single *virtual machine* which is emulated on the actual devices. Prototype coding and testing has shown that this approach is feasible and provides good run-time performance, even on an 8051 CPU.

Each target contains a *kernel* consisting of a standardized set of functions whose CPU-specific implementation is optimized for that specific platform. This kernel supports standard *libraries* and *terminal programs* and *applications* which are written in high-level code for the virtual machine and will therefore run on any terminal with a standard OTA kernel.

### **Virtual Machine**

The software in every OTA terminal is written in terms of a common virtual machine. This is a theoretical 32-bit microprocessor with standard characteristics defining addressing modes, stack usage, register usage, address space, etc. The kernel for each particular CPU type is written to make that processor emulate the virtual machine. The virtual machine concept makes a high degree of standardization possible across widely varying CPU types, and simplifies program portability, testing, and certification issues.

Virtual machine emulation may be accomplished by one of three methods: interpreting tokens representing VM instructions (like Java), translating these tokens into a directly executable "threaded code" form (like Open Firmware), or translating them into actual code for the target CPU. The latter two methods offer improved performance at a modest cost in added target complexity. Present systems employ interpretation.

### **Kernel**

A kernel contains all functions whose implementation depends upon a particular platform (CPU and OS). It includes a selected subset of ANS Forth words, plus a number of specialized OTA functions such as terminal I/O support, token loader/interpreter support, and operations designed to support the particular needs of payment programs. Since it cannot itself be tokenized and downloaded, but must be physically installed (e.g. in PROM), the kernel is intended to be installed once, and not changed thereafter during the lifetime of the terminal. Therefore its functions are carefully designed to be very general in nature and as complete as possible, in order to support a wide range of present and future terminal programs and applications.

A kernel is normally provided by the manufacturer of the terminal in which it resides. It is developed and certified according to the OTA Kernel Specification. Standard kernel functions not appropriate to a particular terminal type (e.g., action of a non-existent device) may be coded for that terminal as null functions, so every kernel has an identical set of functions and the testing and certification process is simplified.

### **Libraries**

OTA Libraries contain higher-level functions that support common features of terminal programs, such as language selection, and common features of applications, such as PIN verification. A terminal may contain several libraries, some accessible to all applications, and some restricted to particular applications or payment systems. Libraries are written and tokenized for the virtual machine, using functions provided in the kernel, and therefore can be run on any terminal.



### **Terminal Program**

A terminal program is part of the Terminal Resident Services and consists of the high-level “personality” characteristic of this terminal type (POS, ATM, etc.). This includes the functions common to all transactions (e.g., card initialization and language selection) as well as the user interface required to select an application and process a transaction. The terminal program at the highest level is typically triggered by a card insertion. Like libraries, a terminal program is written for the virtual machine, based on both kernel and library functions, and supplied in token form. It can therefore be run on any terminal of the appropriate type (e.g., ATM), and is easily changed by downloading over a network at any time.

### **Applications**

A terminal transaction will select an application as part of its processing flow. Applications fall into three general areas: cashless “purses” (Pay Before), debit cards (Pay Now) and credit cards (Pay Later); applications will generally vary in their method of processing a given transaction. Versions of these applications may be provided by different payment systems and further customized by individual issuers or acquirers. Applications are supplied in token form via the communications path, and (if security considerations permit) may be enhanced by token programs on an ICC.

### **Token Compilers and The Token Loader/Interpreter**

Libraries, applications, and terminal programs are written in high-level code for the virtual machine. The OTA development system includes Forth and C compilers for this virtual machine, whose output consists of *tokens*. Tokens may be thought of as machine language instructions for the virtual machine. Tokens are either one or two bytes in length, and therefore represent the program in a form that is both CPU-independent and extremely compact (far more so, for example, than compressed source text).

Each kernel contains a *Token Loader/Interpreter* (TLI), which processes a stream of tokens into an executable form. Once the kernel is installed in a terminal, the libraries, applications, and terminal programs can be downloaded into the terminal in a variety of ways (direct connection to an OTA development host, acquirer hard-wired network, modem and dial-up telephone line, etc.). Program modifications and new applications are downloadable in the same manner whenever needed.

### **ICC Functions**

One function of ICCs is to improve transaction security by incorporating and managing encrypted data and participating actively in the transaction validation process. It is conceptually possible to go beyond these functions and provide for ICCs that also contain program code to enhance a terminal’s transaction processing, thereby providing new opportunities for payment products and services. To facilitate this, “sockets” are provided, that can be “plugged” by issuer-specific functions such as loyalty programs, which may be invoked at appropriate points in the transaction processing. The OTA does not currently propose that ICCs contain entire applications, but only plug behaviors that enhance existing terminal applications.

As far as security is concerned, the presumption is that if an ICC passes the decryption and data authentication tests, whatever functions and plug behaviors are on the card have been certified and are syntactically valid. The terminal decides to allow or disallow the card’s proposed actions only as controlled by the terminal access security functions.

### **Open Terminal Architecture Features**

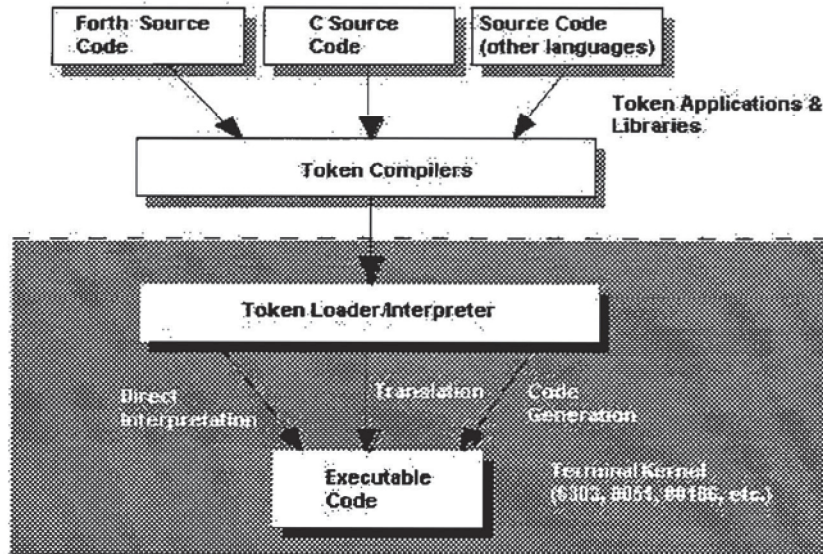
The OTA token set provides program portability in systems with multiple CPU types by passing source code programs of various types through an intermediate compiler whose output is a string of tokens. Target terminals then process this code either by further compiling it to native code, translating it to a different form, or by interpreting it directly. Figure 2 illustrates this process.

The OTA token set covers three main areas. The first is the instruction set of a theoretical processor (virtual machine), and thus provides the instructions necessary for the efficient execution of programs. The second allows what are normally called “operating system functions” to be defined. In the embedded systems for which OTA is targeted, system functions cover not only OTA functions such as communications, TLVs, and library functions, but



also access between program modules and the associated access control mechanism. The third token group includes tokens specific to OTA processing, including such functions as user message management, database management and handling of TLVs (“Tag, Length, Value” data formats, specified by ISO/IEC 8825, and used for communicating with the chip cards).

The OTA token set has been optimized for use on small terminals with ease of compilation, ease of interpretation, and good code density.



**Figure 2**

Tokens in the Interpreter Concept They may be generated from a variety of source code formats, downloaded to a terminal, and interpreted or executed.

### **Virtual Machine CPU Features**

The OTA virtual machine is based on a two-stack architecture, as seen in Forth. This architecture has been further modified for portability, code density, ease of compilation, and for use with other programming languages. For example, it contains provisions for local variables and frame pointers used in C. Thus, OTA token compilers can be written not only for Forth but also C and other languages. It is a byte-addressed, 32-bit machine, with 32-bit registers and stack elements.

OTA defines a single address space for programs. This address space is accessible for data storage only. Programs may not assume that executable code is in this address space. Depending on the actual CPU and kernel implementation strategies the executable code may be in a different address space, or may be under the control of a memory management unit. In any case, programs are not permitted to access their own program memory directly, and any attempt to do so will be flagged during the program certification procedure.

“External memory” may be used to store databases and program modules; however this space is controlled entirely by the kernel, and access to data therein is provided to client programs only indirectly. This memory may be in mapped pages, flash ROM, disk, or simply more main memory — this is completely at the discretion of the kernel implementor and is transparent to the programs.

### **Programs and Tokens**

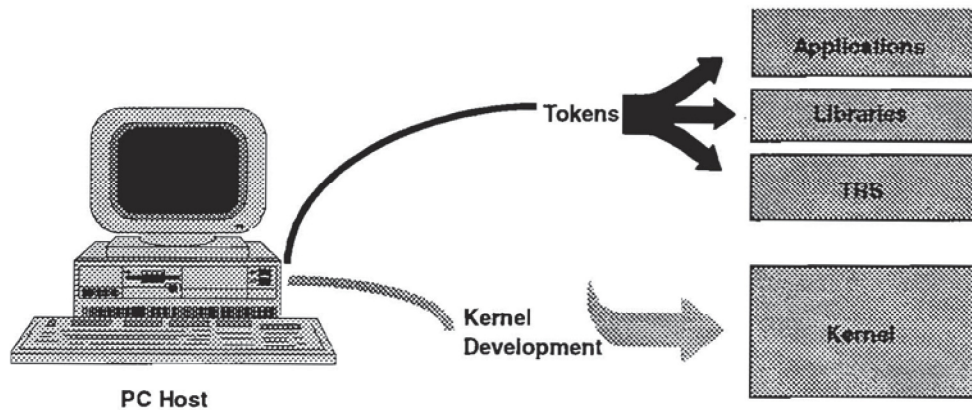
The instruction set of the virtual machine is coded as a byte stream of tokens. The most common functions, including most Forth primitives, are expressed in one-byte or “primary” tokens. Some system functions and most OTA-specific functions are two-byte or “secondary” tokens. Some tokens also have associated values, for such things as literal values and branch offsets.

The token compiler compiles source code as a string of tokens which may be downloaded to the target terminal. After downloading, the terminal uses a token loader/interpreter to execute the tokens.

### **Program and Library Management**

The software on an OTA terminal is organized as a set of separate *modules*. A module is a collection of definitions passed through the token compiler as a single unit. The main Terminal Program, each application, and each library are examples of modules.

Modules need to communicate to perform overall terminal functions. A program module, for example a library, contains a number of definitions. Some of these are used only internally, and others must be accessible externally. Following all the definitions in a module there is a special section where definitions to be referred to externally are "exported" or listed in a way that constructs a table of entry points published for access by higher-level modules.



**Figure 3**

### **Architecture Summary and Status**

The basic architecture of OTA was successfully validated in the prototypes exhibited at the Europay Members' Meeting in Seville. Issues to be revisited include placement of certain functions in the kernel vs. external token programs, minimum and maximum size requirements, and further definition of standard libraries and application functions. In addition, intensive efforts will be directed in the next few months toward development of salable software products, including development kits and validation suites for both kernels and token modules.

# EXHIBIT E



- 1 Kirillin, Viacheslav  
JBS  
59 Bolshoi Dr. P8  
St. Petersburg **Russia**  
W(812)233-3410 F(812)319-9709
- 2 Anway, Allen  
A2D2 Electronics  
1219 North 21st St.  
Superior, WI 54880-5057  
W/H(715)394-4061
- 3 Bean, Warren Louis  
9804 Lonsdale Dr.  
Austin, TX 78729  
(512)257-1775 F(512)891-7690
- 4 Benson, Steve  
Edith Cowan University  
Churchlands Campus-Pearson St.  
Churchlands, WA 6018 **AUSTRALIA**  
W:09 273 8369 F:09 273 8754
- 5 Bradley, Mitch  
FirmWorks  
480 San Antonio Rd. Ste. 115  
Mountain View, CA 94040-1218  
W(415)917-0100
- 6 Brumm, Doug  
Michigan Technological Univ.  
EE Dept. - Michigan Tech  
Houghton, MI 49931  
W(906)487-2515 F(906)487-2949
- 7 Carpenter, John D.  
Stanford Univ. - Genetics  
855 California Ave.  
Palo Alto, CA 94304  
W(415)812-2006 F(415)812-1975
- 8 Carter, Everett F. (Skip)  
Taygeta Scientific, Inc.  
1340 Munras Ave. Ste. 314  
Monterey, CA 93955  
W(408)641-0645 F(408)641-0647
- 9 Chapin, Elliott  
24 Monteith Street  
Toronto, Ontario M4Y 1K7  
(416)921-9560 **CANADA**
- 10 Chapman, Robert  
New Micros, Inc.  
10807 Castle Downs Rd. #209  
Edmonton, Alberta T5X 3N7  
W(403)413-7299 **CANADA**
- 11 Dumse, Randy  
New Micros, Inc.  
1601 Chalk Hill Rd.  
Dallas, TX 75212-5804  
W(214)339-2204 F(214)339-1539
- 12 Falat, Edward  
30 Janet Rd.  
Meriden, CT 06450-4828  
W(203)929-8431 ext329 F(203)929-3898
- 13 Forsley, Brenda J.G.  
The Forth Institute  
70 Elmwood Ave.  
Rochester, NY 14611  
W/F(716)235-0168
- 14 Forsley, Lawrence P.G.  
JWK International - Ste. 1000  
7617 Little River Tpk  
Annandale, VA 22003  
W(703)750-0500
- 15 Fox, Brian  
Microtronix Systems Ltd.  
200 Aberdeen Dr.  
London, ON **CANADA** N5V 4N2  
W:(519)659-9500 F(519)659-8500
- 16 Fox, Jeff  
Ultra Technology, Inc. - (ITV Corp)  
2510 10th St.  
Berkeley, CA 94710  
(510)848-2149
- 17 Frenger, Paul  
814 Silvergate Dr.  
Houston, TX 77079-5001  
W(713)293-9484 F(713)293-9446
- 18 Hall, John  
4800 Allendale Ave.  
Oakland, CA 94619  
W(415)786-6927 (510)534-7074
- 19 Haydon, Glen B.  
Mountain View Press/WISC Technologies, Inc.  
Rt 2 Box 429 / 19500 Skyline Blvd.  
La Honda, CA 94020  
(415)747-0760
- 20 Hodson, Bernard  
PO Box 3310  
Ottawa, **CANADA** K1P 6H8  
W(613)523-5322 F(613)523-5322
- 21 Hooley, David  
ACI Systems  
424 Myer Terrace  
Leola, PA 17540-1217  
W(717)656-4535
- 22 Johannes, Peter  
EPSS  
198A Chaussee de Tervuren  
B-1410 Waterloo, **BELGIUM**  
W:32 2 352 5256 F:32 2 352 5839
- 23 Kaeufer, Edward  
CPMI  
767 Samish Island Rd.  
Bow, WA 98232  
W/F(360)766-8319
- 24 Kupisz, Ken  
14 Mill Cove  
Toronto, Ontario M8X 2S6  
(416)239-0543 **CANADA**
- 25 Leberknight, Randy  
Motorola, Inc.  
2900 S. Diablo Way MS DW 278  
Tempe, AZ 85282  
W:(602)438-3275 F:(602)438-3836
- 26 Lee, Jon  
MicroProcessor Engineering Ltd.  
133 Hill Lane  
Southampton, **ENGLAND** SO1 5AF  
W:44-703-631441 F:44 703 339 691
- 27 Louis, Dr. Gregory  
Dynamicro Consulting Ltd.  
282 Waratah Ave.  
Newmarket, Ontario L3Y 5J2 **CANADA**  
W(905)738-3741 F(905)738-3712

- 28 Lowry, Dave  
Honeywell Technology Center  
18280 Tristram Way  
Eden Prairie, MN 55346  
(612)951-7294
- 29 Mason, Dave  
Ryerson Univ./Comp. Sci  
Rogers Comm. Bldg  
Toronto, Ontario CANADA  
(416)461-2112
- 30 McCracken, Ken  
390 Jones St.  
Toronto, Ontario M4J 3G3  
(416)466-3553 CANADA
- 31 McDonald, Robert  
DSX Embedded Programming  
872-25th St. E.  
Owen Sound, Ontario N4K 6T4  
(519)372-9916 CANADA
- 32 McGowan, Marty  
24 Herning Ave.  
Cranford, NJ 07016-1946  
(908)582-3464
- 33 Melton, David  
Cellular Technical Services, Inc.  
2401 Fourth Ave. Ste. 808  
Seattle, WA 98121  
W(206)727-0748 F(206)727-1262
- 34 Moore, Chuck  
Computer Cowboys  
410 Star Hill Rd.  
Woodside, CA 94062  
(415)851-4362
- 35 Nieuwenhuijzen, Hans  
Space Research Organiz. Netherlands  
Sorbonnelaan 2  
NL-3584 CA Utrecht, THE NETHERLANDS  
W(31)30-2535723 F(31)30-2540860
- 36 Pelc, Stephen  
MicroProcessor Engineering Ltd.  
133 Hill Lane  
Southampton, ENGLAND SO1 5AF  
W:44-703 631441 F:44 703 339 691
- 37 Pfau, Ewald  
Honigfal 145  
8301 Lassnitzhohe, AUSTRIA  
43-3133-2813
- 38 Prentice, Lloyd  
Prentice Associates, Inc.  
160 Tea Rock Lane  
Marshfield, MA 02050  
W(617)834-8002 F(617)834-8009 ext.40
- 39 Rather, Elizabeth  
FORTH Inc.  
111 N. Sepulveda Blvd. #300  
Manhattan Beach, CA 90266  
W(310)372-8493 F(310)318-7130
- 40 Rible, John  
Quicksand  
317 California St.  
Santa Cruz, CA 95060-4215  
W/H(408)458-0399 F(408)662-0807
- 41 Rodriguez, Bradford J.  
T-Recursive Technology  
115 First St. Ste. 105  
Collingwood, Ontario L9Y 4W3  
(519)986-4266 CANADA
- 42 Solntseff, Nicholas  
McMaster University  
Dept. of CS and Systems  
Hamilton, Ontario L8S 4K1 CANADA  
W(905)525-9140 x23443 F(905)627-8395
- 43 St-Laurent, Graham
- 44 St-Laurent, Ray  
Fredericton, NB E3B 6Z9  
H:(506)459-4827 CANADA
- 45 Thomas, Jet  
340 Calle Miramar  
Redondo Beach, CA 90277  
(310)373-5488
- 46 Ting, C.H.  
OFFETE  
1306 S. B St.  
San Mateo, CA 94402  
W(415)574-8250 F(415)572-2743/(415)571-5004
- 47 Verne, J.D.  
56-d Kensington Ave.  
Toronto, Ontario M5T 2K1 CANADA
- 48 Wagner, Richard  
356 Ruxton Ave. Apt. B  
Manitou Springs, CO 80829  
W/H(719)685-0152
- 49 Zettel, Leonard  
LenZ Computer Consulting  
652 B S. Brady  
Dearborn, MI 48124-2336  
W(313)592-2773 F(313)592-2381
- 50 Ziolkowski, Robin  
ALEX:Creative Solutions in Technology  
85 Major Mackenzie Dr. East  
Richmond Hill, ON L4C 1G8  
(905)508-6738 CANADA