Definitions
Institute for Applied Forth Research, Inc.
70 Elmwood Ave.
Rochester, NY 14611

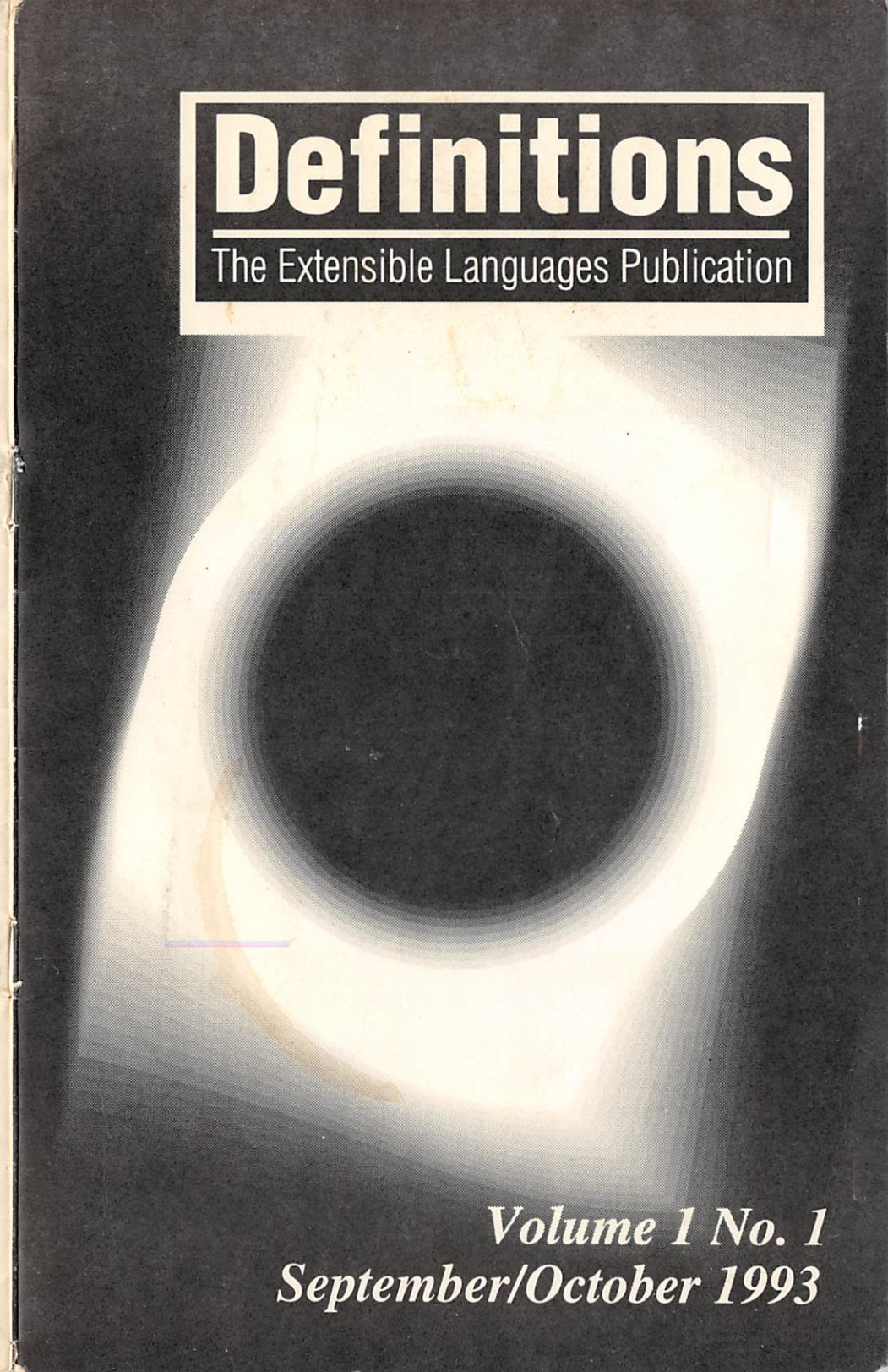
Bulk Rate
U.S. Postage
PAID
Rochester, NY
Permit No. 50

Korsak,Andrew

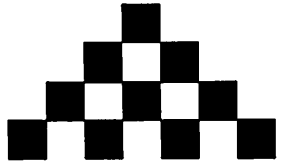
504 Lakemead Way
Redwood City, CA
94062

Definitions

The Extensible Languages Publication



Volume 1 No. 1
September/October 1993



SILICON COMPOSERS INC

The SC32 Line-Up

SC32[tm] 32-bit Forth Microprocessor
- 8 or 10 MHz operation.
- 1-cycle instruction execution.
- Multiple Forth words per clock cycle.
- Contiguous 16 GB data and 2 GB code space.
- Stack depths limited only by available memory.

SBC32 (Single Board Computer32)
- 64 KB to 512 KB 0-wait-state static RAM.
- 100mm x 160mm Eurocard size board.
- SC/Forth32 Forth-83 based system included.

PCS32 (Parallel Coprocessor System32)
- Full-length PC/XT/AT plug-in board.
- 64 KB to 1 MB 0-wait-state static RAM.
- SC/Forth32 Forth-83 based system included.

DRAMIO32 (DRAM, SCSI, Serial, Parallel)
- Eurocard size, plugs onto SBC32 or PCS32.
- Holds up to 16MB on-board DRAM.
- 16-bit parallel, 4 serial ports, time/date.
- Interfaces up to 7 SCSI devices.
- Software drivers in source code form.

FAD32 (A/D Converter, WatchDog Timer)
- Eurocard size, plugs onto SBC32 or PCS32.
- 16 channels, 12 msec, 12 bit plus sign A/D.
- Voltage generation, wdog timer, proto area.
- Software drivers in source code form.

655 W. Evelyn Ave#7, Mtn View, CA 94041 (415) 961-8778

ENCODER PRODUCTS COMPANY

The Responsive People in Controls

Synergy Plus - An Intelligent Distributed Control System

+ Hardware

AIO 1218 - 8 inputs, 2 outputs, 16 digital I/O, PID routine
DIO 1248 - 48 digital I/O, +5 vdc logic
CTR 1224 - 2 high speed 24 bit counters, 16 digital I/O, 3 modes
SIO 1232 - 2 serial I/O, 16 digital I/O, RS-232/RS-485 options
RCM 1244 - Byte FIFO interface to IBM (Forth, Lotus, Windows)
* MCB 1290 - 2 axes of servo control, slaving, fractional ratioing

* MCB 1290 plugs directly to any baseboard module via SBX
All modules include SBX interface, Bitbus communications, Diagnostic LEDs, Watchdog, and many more standard features.

+ Software

FREE!

Multitasking operating system (DCX pre-emptive)
Resident interactive programming kernel (Forth-83)
Resident development interface and editor
Disk-based library of application utilities
- floating point arithmetic
- dynamic memory management
- state language programming
- message passing
- object oriented programming
- many more available for your unique application

+ Integration Services - Turn-key Systems

System specification
System design
Programming
Installation

1601B Highway #2 • Sandpoint, ID 83864

Call 800-366-5412

In Postscript these would be entered as follows:

/DROP {pop} def
/SWAP {exch} def
/PICK {index} def
/DEPTH {count} def

This brings us naturally to the next subject of discussion: PostScript definitions. In order to cover them, however, some interpreter details must also be covered.

Forth's syntax is remarkably consistent. Everything is a word or a number, and these are separated by spaces. The <CR> terminates a line. Other than that, there are no escape characters. Individual words may have a syntax associated with them. The comment operator "(", for instance, parses a comment out of the input stream, but the Outer Interpreter itself is pristinely clean. It is a terrible temptation to "improve" this by adding characters with special meaning. Most Forth vendors have resisted.

PostScript, however, approaches this issue from a conventional programming language stance - no syntax style is sacred. Starting with the basic Forth idea additions are made. The **nul tab <LF> <CR>** and **space** are considered white space. Unfortunately, exceptions often lead to more exceptions. The **( ) < > [ ] { } / %** are special characters. All others are considered "regular" characters. The **( )** pair indicate a string. So does a **/** , but only one without any white space, called a "name". The **{ }** pair indicate a procedure. Notice white space around these special characters is not critical, since they are control characters, rather than the names of routines.

Given this brief introduction to the PostScript syntax, the definition structure shown above can be explained. Starting with **DROP**, the **/** indicates a string follows, in this case the name of a routine about to be defined. The **{ }** brackets a procedure which is put on the stack, in this case the procedure calling a **pop**. Finally the defining word **def** takes the procedure off the stack, the name next down on the stack, and assigns adds a definition to the user's dictionary. This should have a very familiar ring to those who have programmed in Forth, even though the details are slightly different.

Occasionally PostScript seems more Forth like than Forth itself. Forth does not explicitly handle strings. This is left to individual words to accomplish, such as "(" mentioned earlier. The colon, :, defining word in Forth appears to work in prefix, since the colon comes first followed by the name. If Forth had a string stack it could handle all such matters in postfix. PostScript, on the other hand, seems to remain true to the postfix style. Its object orientation provides for many kinds of things to be on the stack, including in the example above, strings.

Another place this shows up in PostScript is conditionals. While Forth uses program control branching to handle conditionals, PostScript uses postfix stack operations. The **if** operator in PostScript takes a procedure from the stack and evaluates a Boolean below it. If the Boolean is true the procedure is executed. If not, it is dropped. This is far more consistent than branched program flow.

Next time, sample PostScript code, and a real application.

-R. D. Dumse

Definitions

Volume 1 No. 1
September/October 1993

Definitions is published 6 times a year

Table of Contents

What is Forth technology?
How do I program my laser printer in Postscript?
What language really works for rapid prototyping?
How do I find consultants or programmers?
Who really understands hardware and software?
How can I make Microsoft Windows work for me?
What works in real time for embedded systems?

These and more questions are answered in **Definitions**. Subscribe Today!

Subscriptions are $25/year in North America, and $30/year outside North America.

Definitions is distributed bi-monthly by the Institute for Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, NY, 14611. Definitions is published and edited by Lawrence P. G. Forsley. Circulation and advertising is handled by Brenda J. G. Forsley. Send inquiries to us at that address, or at our phone: (716)-235-0168

Journal of Forth Application and Research *(JFAR)*

JFAR is the only refereed journal devoted to Forth-like languages. Special topic issues have included object oriented programming and real-time expert systems. *Call for a complimentary copy!*

JFAR Issue	#1	#2	#3	#4
Volume 2		❑	❑	❑
Volume 3	❑	○	❑	❑
Volume 4	❑	○	❑	❑
Volume 5	○	❑	❑	❑

❑ = $15.00 , ○ = $20.00 (Conf Proc)

Full Sets:

Volume 1		❑ $30.
Volume 2	(3 issues)	❑ $40. *Save* $5.
Volume 3	(4 issues)	❑ $50. *Save* $15.
Volume 4	(4 issues)	❑ $50. *Save* $15.
Volume 5	(4 issues)	❑ $50. *Save* $15.
Volumes 1-4		❑ $170. *Save* $35.
Volumes 1-5		❑ $220. *Save* $50.

JFAR Subscriptions:

Volume 6
- ❑ $60.00 Individual, USA
- ❑ $65.00 Individual, North America
- ❑ $75.00 Individual, Europe/Asia
- ❑ $145.00 Corporate, North America
- ❑ $160.00 Corporate, Europe/Asia

Note: No additional shipping charges on Subscriptions.

Order Form:

Name_____

Address:_____

_____

City:_____

State:_____ ZIP: _____

Phone Work:_____
Home: _____
FAX: _____

Rochester Forth Conference Proceedings

The international Rochester Forth Applications Conference has been held at the University of Rochester for the past 13 years and has been attended by as many as 175 people from around the world. Most Conference Proceedings consist of long papers by invited speakers addressing the Conference theme and as many as 50 shorter papers on all topics of Forth application and implementation.

- ❑ $25. 1981 *Forth Standards*
- ❑ $25. 1982 *Databases & Process Control*
- ❑ $25. 1984 *Real Time Systems*
- ❑ $20. (Vol.3,#2)1985 *Software Productivity*
- ❑ $20. (Vol.4,#2)1986 *Real Time AI*
- ❑ $20. (Vol.5,#1)1987 *Comp Architecture*
- ❑ $25. 1988 *Prog Environments*
- ❑ $25. 1989 *Industrial Automation*
- ❑ $25. 1990 *Embedded System*
- ❑ $30. 1991 *Automated Instruments*
- ❑ $30. 1992 *Biomedical Applications*
- ❑ $30. 1993 *Process Control (fall '93)*

Set of any 4 or more Proceedings
Save $5./Proc., up to $60.!

- ❑ $20.00 Forth Ref Bibliography, 3rd Ed
- ❑ $25.00 1988 ASYST Conference Proc
- ❑ $62.00 *Stack Computers: the New Wave*
  By: Dr. Philip J. Koopman, Jr.
- ❑ $50.00 *Scientific Forth*
  By: Dr. Julian Noble(with software on disk)

Purchase Subtotal: _____
Shipping Subtotal: _____
($5./book,15 Max. North America)

Grand Total: _____
❑ MC ❑ VISA ❑ Check
Card #_____
Expiration Date: _____

Institute for Applied Forth Research, Inc.
70 Elmwood Avenue
Rochester, NY 14611
(716)-235-0168 voice (716)-328-6426 fax
72050.2111@compuserve.com email

PostScript vs. Forth

How similar are PostScript, the page description language, and Forth? Surprisingly so. Both languages fall well into the category of languages covered by Definitions. This column will be a brief tutorial comparing the similarity of the two languages. A familiarity of Forth is assumed.

In the Postscript Language Reference Manual, Second Edition, is written, "the PostScript language builds on elements and ideas from several of the great programming languages. The syntax most closely resembles that of the programming language FORTH." In the First Edition the authors were perhaps a bit less comfortable with naming their heritage. In the opening preface they said,

Although the Design System language and its successors bear a superficial resemblance to the FORTH programming language, their conception and development were entirely independent of FORTH

This disclaimer is completely missing in the Second Edition.

While I am not a Postscript expert, I have now designed several logos and special effects for my company. I found it only took two weeks to make the transition from someone who had heard of Postscript, to someone who was a functional programmer.

In this issue of Definitions, I would like to present some of the basic words that are similar between the two languages. Later issues will cover more detail.

Both languages are postfix, have dictionaries, make named definitions, and have an interactive outer interpreter. To see the similarities, a good place to start will be looking at the stack operators.

Postscript offers eleven Operand Stack Manipulation Operators: **pop exch dup copy index roll clear count mark cleartomark** and **counttomark**. The FORTH-83 Standard has nine stack manipulation words in the Required Word Set: **?DUP DEPTH DROP DUP OVER PICK ROLL ROT SWAP**. While the stacks of Postscript have data typing properties, the similarities are obvious.

Two sets of words have the same name, save capitalization: **dup** and **DUP**, and, **roll** and **ROLL**. Indeed, **dup** and **DUP** have exactly the same meaning in both languages - the top item on the data stack is duplicated. The words **roll** and **ROLL** do not have exactly the same meanings. The PostScript **roll** is a counted roll, rolling up n elements j times. The Forth **ROLL** brings up the n-th element one time. Still, the underlying idea is similar in both.

Other words have identical functions, but different names: **pop** and **DROP**, **exch** and **SWAP**, **index** and **PICK**, **count** and **DEPTH**.

It is easy to imagine redefining the names of one language so the user of the other could easily cope with more familiar names. In Forth this would be:

```
: pop DROP ;
: exch SWAP ;
: index PICK ;
: count DEPTH ;
```

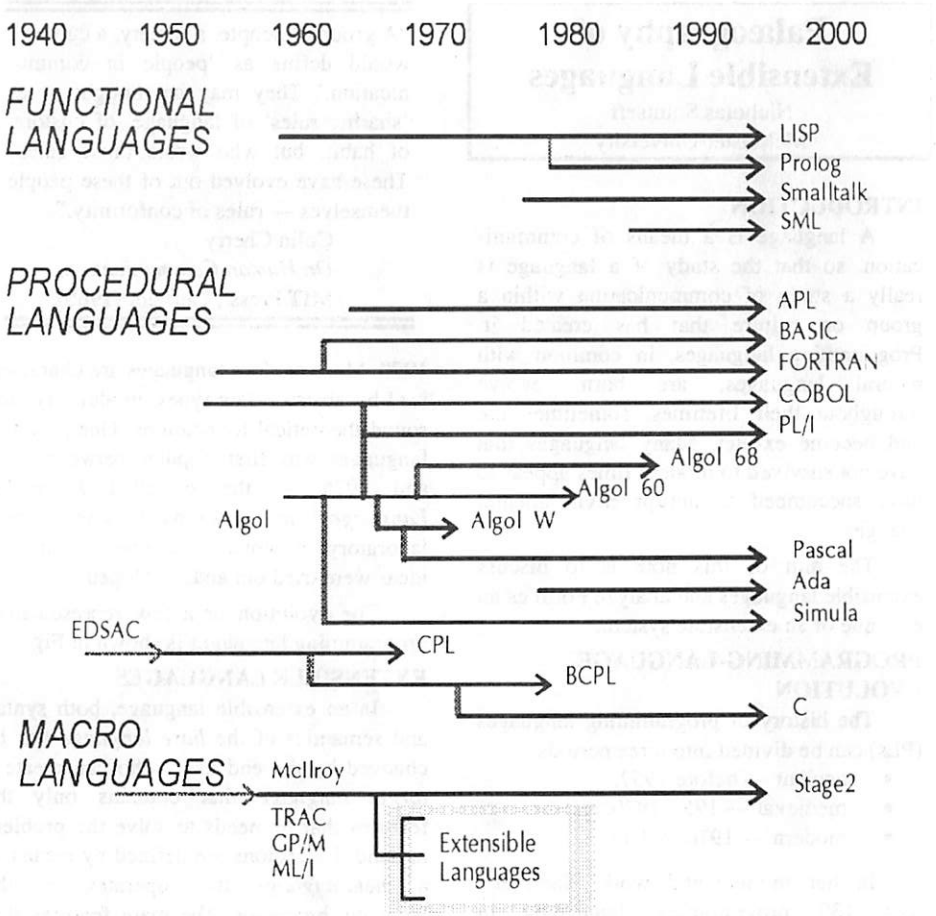


Figure 1. The evolution of some representative programming languages.

Extensible languages arose from the desire of their developers to create small, efficient, and adaptable programming systems in response to the introduction of such omnibus languages as PL/I which are meant to contain everything that any programmer might need. In other words, they can be viewed as an attempt to resolve the perennial controversy between proponents of large general-purpose and small special-purpose languages.

Extensible languages appeared in large numbers in the late 1960s and early 1970s — Solntseff and Yezerski (*op. cit.*) list over a hundred in their review — and were aimed at facilitating language extension through the addition of:

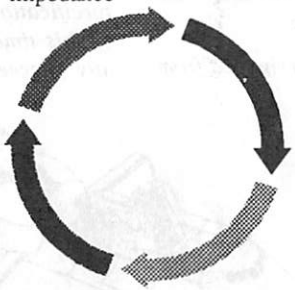
- new forms of access to data
- new forms of statements
- new control-flow constructs.

(To be Continued in next issue)

Definitions
the extensible languages publication

Definitions covers the unique world of extensible computer languages. Nowhere else does such a diverse group of computer technologies appear. More than detailing the languages, ***Definitions*** is dedicated to the underlying technology.

Extensible languges at their very core are fundamentally different than other computer languages. Extensible languages provide a good impedance match between how we think and how they represent knowledge. Like the very DNA in our cells, extensible languages employ interpreters which interpret themselves.



As such, these languages are the key to computing in the 21st Century. For several years Adobe's Postscript™ has provided a device independent page description language. Similarly, Sun Microsystems has developed and used a device and processor independent Forth known as Open Boot on nearly *1 million workstations.* Recently, Philips announced their Interactive CD, which has a Forth-based authoring system available. Eastman Kodak uses the Philips disc player technology for their new Photo CD. If you look under the hood, an increasing number of products are *Definition* based.

The survival and prosperity of programmers and engineers will increasingly depend upon extensible languages and the companies who provide them. ***Definitions*** binds together this diverse community by opening lines of communication, searching out applications,and providing a forum for our success.

Whether polyFORTH™ in space on NASA shuttle missions, or Open Boot in Sun Workstations; whether in Europe, Russia or China: *we're there to report for you* providing a comprehensive overview of Extensible languages like Keithley's ASYST™, for science and engineering; Opto 22's Cyrano™ for industrial control; FORTH Inc.'s Express, for process control; and Adobe's Postscript for page layout. ***Definitions*** offers you a chance to keep up with and participate in the latest developments in this field. ***Definitions*** is there *now.*

Join us!

Subscriptions are $25 year in North America, and $30./year outside North America.

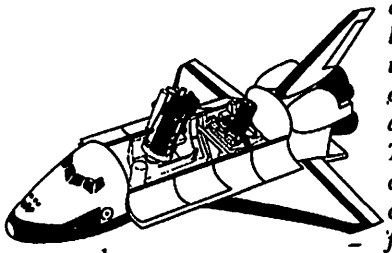
Name:_____
Address:_____
_____
Phone: _____
Payment: Payment:❑ **check**
❑ **MC** ❑ **VISA**
ACCT#_____
Expires: _____

Definitions *is distributed bi-monthly by the Institute for Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, NY, 14611. Definitions is published and edited by Lawrence P. G. Forsley. Circulation and advertising is handled by Brenda J. G. Forsley. Send inquiries to us at that address, or at our phone: (716)-235-0168*

Observation
Feynman on Top Down Design

Top down neglects the need to play with a problem, determining its dimensions and boundaries. The late Dr. Richard Feynman, Physics Nobel laureate, was a member of the Rogers commission which investigated the Challenger disaster. Feynman's review of the failure of a top down strategy, both technically and in management, for the shuttle suggests similar difficulties with software. He noted that unlike the shuttle,

Most airplanes are designed "from the bottom up," with parts that have already been extensively tested. The shuttle, however, was designed "from the top down"--to save time. But whenever a problem was discovered, a lot of redesigning was required in order to fix it.[1]



The final Presidential committee report to NASA identified both the O-Ring failure in the solid fuel booster and a breakdown in management communication as the causes of the Challenger disaster. In an appendix to that report Feynman looked at other subsystems, like the shuttle's three main engines:

The usual way that such engines are designed (for military or civilian aircraft) may be called the component system, or bottom-up design. First it is necessary to thoroughly understand the properties and limitations of the materials to be used (turbine blades, for example), and tests are begun in experimental rigs to determine those. With this knowledge, larger component parts (such as bearings) are designed and tested individually. As deficiencies and design errors are noted they are corrected and verified with further testing. Since one tests only parts at a time, these tests and modifications are not overly expensive. Finally one works up to the final design of the entire engine, to the necessary specifications. There is a good chance, by this time, that the engine will generally succeed, or that any failures are easily isolated and analyzed because the failure modes, limitations of materials, et cetera, are so well understood. There is a very good chance that the modifications to get around final difficulties in the engine are not very hard to make, for most of the serious problems have already been discovered and dealt with in the earlier, less expensive stages of the process.

The space shuttle main engine was handled in a different manner--top down, we might say. The engine was designed and put together all at once with relatively little detailed preliminary study of the materials and components. But now, when troubles are found in bearings, turbine blades, coolant pipes, et cetera, it is more expensive and difficult to discover the causes and make changes. ... Using the completed engine as a test bed ... is extremely expensive. One does not

[1]Quotations from Feynman, Richard. P., *"What do you care what other people think?", Further Adventures of a Curious Character.*

Palæography of Extensible Languages

Nicholas Solntseff
McMaster University

"A group of people, a society, a culture I would define as 'people in communication.' They may be thought of as 'sharing rules' of language, of custom, of habit; but who wrote those rules? These have evolved out of these people themselves — rules of conformity."

Colin Cherry
On Human Communication,
MIT Press , 2nd. Ed. (1965).

INTRODUCTION

A language is a means of communication, so that the study of a language is really a study of communication within a group or culture that has created it. Programming languages, in common with natural languages, are born, evolve throughout their lifetimes, sometimes die and become extinct. Many languages that have not survived to modern times appear to have succumbed to abrupt environmental changes.

The aim of this note is to discuss extensible languages and analyze Forth as an example of an extensible system.

PROGRAMMING-LANGUAGE EVOLUTION

The history of programming languages (PLs) can be divided into three periods:

- ancient — before 1957,
- medieval — 1957-1970, and
- modern — 1970 to date.

In her monumental work, Sammet[1], lists 130 programming languages in existence before 1957. With few exceptions (notably COBOL and FORTRAN) these were various autocoders, assembler-like languages, or macro processors. Most were the result of research into *Automatic Programming*, the development of tools for computer-aided creation of programs.

The widely used programming languages of today — BASIC, C, COBOL, LISP, and Pascal, have their origins around 1970. More modern languages are characterized by abstract data types, modularity, and sound theoretical foundations. One group of languages was first popular between 1965 and 1975 — the so-called *Extensible Languages*[2]. In a real sense they served as a laboratory in which a number of modern ideas were tried out and developed.

The evolution of a few representative programming languages is shown in Fig. 1

EXTENSIBLE LANGUAGES

In an extensible language, both syntax and semantics of the *base language* can be changed by the end user, who can create a *target language* that contains only the features that he needs to solve the problem at hand. Extensions are defined by means of a *metalanguage* that operates on the language processor. The main features that distinguish extensible languages from conventional ones are:

- symbol tables are not discarded after the lexical phase
- the compiler can be invoked at run time
- some or all phases of compilation can be changed by means of the metalanguage.

[1]J. Sammet, *Programming Languages: History and Fundamentals*, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1969)

[2]N. Solntseff and A. Yezerski, "A survey of extensible programming languages," in *Annual Survey of Automatic Programming*, Vol. 7, (part 5, 1974), pp. 267-307.

limited memory without an operating system, were most frequently mentioned. Another "crippled" environment is a workstation during the boot process. The Sun Sparc station boots into a variant of Forth before launching Unix. This system allows developers and system administrators to talk to hardware in ways which would be impossible once UNIX is launched. Printers running Postscript represent another "crippled", embedded environment where threaded, interpreted languages have proven successful. A fourth area where Forth is successful involves systems which attempt to push hardware to the absolute limit such as specialized imaging systems. Here Forth chips can prove competitive. Again, systems which push hardware to the limit in one area, such as imaging, usually do so at the expense of other services. They offer only rudimentary operating systems and other services. Again this represents a "crippled" environment.

The consensus was that for large projects developed and running on workstations or personal computers, C and C++ represent a superior environment. This conclusion suggests that Forth has lost the main stream to alternative languages. Given Forth's natural advantages of interactiveness, extremely rapid incremental development and interpretive ability to examine structures, it is surprising that Forth is not better accepted as a mainstream language.

Next Issue: Forth's tradeoffs vs. C and C++.

wish to lose entire engines in order to find out where and how failure occurs.

A further disadvantage of the top-down method is that if an understanding of a fault is obtained, a simple fix--such as a new shape for the turbine housing--may be impossible to implement without a redesign of the entire engine.

The current method of software design is also top down design, and top down coding. One is expected to build to a specification and the code flows. Indeed, in large software projects there are software analysts who design, and coders who program. The programming hierarchy goes from specifiers (managers), to designers, to coders, to documentors to maintainers.

Consequently, there is ample room for miscommunication among the many levels. Indeed, the code maintainers may have the clearest idea of what is being done, but this is rarely passed back to the design team: and even if it is, its far too late into the product.

Fortunately, Forth offers an alternative software methodology. One can do top down design and bottom up coding, adopting a methodology of iterative design. Iterative design is even more important where specifications are incomplete or changing, which seems to more often be the norm than the exception. This methodology also allows rapid prototyping.

- by L. P. G. Forsley

Future articles will examine actual software projects.

1993 Rochester Forth Conference on Process Control

The 13th International Rochester Forth Conference was held at the University of Rochester June 23 - 26. The Conference was sponsored by Bradley Forthware, Inc., New Micros, Inc., Miller Microcomputer Services, Inc., and Dash, Find & Associates. Over 30 papers were presented. Among them, author Dr. Julian Noble presented a paper on accident reconstruction. Representatives from Johns Hopkins University Applied Physics Laboratory discussed the development of Forth architecture. NASA focused on the realities of spaceflight and software. FORTH, Inc. demonstrated their latest polyFORTH based-product for process control, EXPRESS.

Working groups concentrated on Forth processors and state machines. This year's annual 4K run had its largest set of runners, strollers and bikers ever. Vendors Day included full hour seminars on Express by Elizabeth Rather at Forth, Inc. and Open Boot by Mitch Bradley, of Bradley Forthware, Inc. A rousing "Forth Feud" trivia night closed the Conference for this year.

The Conference Proceedings will be available in the fall from the Forth Institute. See their order form for details.

4K run, roll and stroll

Forth, C and C++[1]

Twelve years ago, when the Rochester Forth conferences began, typical personal computers were systems powered by 8 bit processors. Storage consisted of two small floppy disk drives and memory was typically 64 Kb at best. In such a system it was difficult to imagine developing with a sophisticated compiled language such as C or Pascal. The major competition for Forth was interpreted BASIC.

Forth and C
Part 1 of 3

Today typical personal computers have as much power as mainframes of that era possessed. Todays personal computer has a 32 bit processor, at least 8 Mb of memory, a relatively fast hard disk holding on the order of 100 MB. Today compiled languages work well in this more powerful environment. Proponents of Forth must argue its merits against powerful competing languages. In addition, modern compiled languages now have debuggers which allow developers to step through the code and browse data structures. The capabilities of a modern browser give C and Pascal developers access to many of the capabilities Forth programmers have enjoyed in interpreted mode.

Modern programs are changing as well. Windowing, graphically based programs have become the norm. These programs are large and sophisticated. It is not unusual to see programs with 500K or more of code requiring 2-3 MB of storage. These applications are an order of magnitude larger than those we would build twelve years ago and raise new issues as to what are the typical design drivers.

Classical developers tried to get maximum performance out of a system with limited resources. Classical systems were limited in terms of memory, disk speed and processor speed. Developers worked to use these limited resources to maximum advantage choosing the best system to maximize their own and their system's capabilities. Today the limiting factor in software development is the developer. While the capabilities of today's systems have increased by orders of magnitude over earlier systems, developers have not gotten any smarter. Today and increasingly in the future the successful systems will be those that maximally aid the developer.

In the near future, we can imagine systems where incremental compilation and linking of C or C++ programs is essentially instantaneous. Already some environments such as Think C on the Macintosh comes close to this goal. In this era, Forth's advantage of incremental linking and immediate execution is greatly reduced. It is time to reexamine the assumptions underlying selection of Forth and ask how Forth will fit into a new era.

Forth was acknowledged to be superior and where C++ was a better system. In collecting a list of Forth success stories, a common theme emerged. Forth was the system of choice in environments that would be considered "crippled" compared to the capabilities of modern personal computers. Embedded systems, functioning on limited hardware, usually

[1] Excerpted from "Working Group on C and C++", *Proceedings of the 1992 Rochester Forth Conference on Biomedical Applications*, pp 113-115. Chaired by Dr. Kent Brothers and written by Horace Simmons.

Consultant Spotlight: Mr. Gary Bergstrom

Gary Bergstrom

Gary has been known to the Forth community for almost ten years through his consulting and the Rochester Forth Conferences. He has an excellent background in analog and digital engineering with over 10 years of Forth programming experience as well as ASYST, Assemblers, C, and QuickBasic. He also has engineering management experience. Gary earned an MSEE in 1977 from Ohio State and a BA in Physics from Wittenburg University. He has one patent.

As a consultant he has specialized in design consulting for analog and non-linear circuits, digital designs including microcomputers and digital signal processing and in software emphasizing data acquisition and real time systems. He has specific instrument, audio, and biomedical applications experience with both hardware and software. He prefers not to work on military projects.

Gary is well versed in F83, MicroProcessor Engineering's Forth, and New Micros' Max-FORTH. He is familiar with several microprocessors including the Intel 80386 family, NEC V25, Motorola 6809, 68HC11 and 56000 DSP.

Gary notes:

I was the manager of software at Orion Research but I still did a lot of the software. I like to get my hands dirty. I'd rather do the work than manage a group of people. My business is structured the same way. I like systems that are small enough for one person to understand the whole system. All one person jobs are much more efficient.

His considerable experience with both Forth and C has led him to observe:

Trying to factor things is difficult in C. You tend to write in monolithic chunks. I try to write in as small a chunk as I can.

The president of New Micros, Inc., Randy Dumse, recommends Gary highly:

Gary developed our V25 Forth, among other things. Generally, when you hire a consultant, it's hit or miss. Most fall far short of the mark. Ocassionally, you'll get about what you pay. On the other hand, I always felt I got more than my money's worth dealing with Gary. He never did less than a first-rate job.

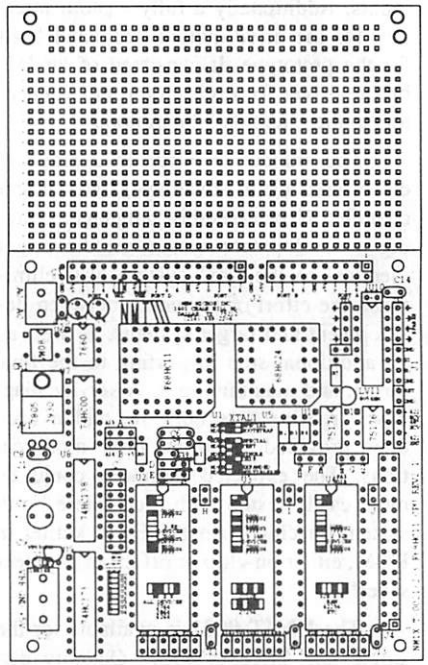
He is available for project or product definition, electronic design, prototype design and construction, manufacturing support and documentation. He is quietly competent and a pleasure to work with!

Consider Gary Bergstrom for your next project!

You can reach him through:
Dash, Find & Associates
70 Elmwood Ave
Rochester, NY 14611
(716)-235-0168

New Micros, Inc.

New Micros, Inc. in Dallas, Texas is hard at work doing yet another custom project. New Micros makes approximately 40% of total sales, the largest single sales category, doing custom systems. Their approach of putting Forth inside single chip computers is unique in the community. As a business strategy it has proved successful. New Micro's sales double every few years.



NMIX /T-0021 Board Layout

New Micros is forming a new partnership with the U.S. arm of Crouzet Corporation, the French founded multinational company, who recently purchased GORDOS. Crouzet is seeking to expand its reach in the single board controller and PLC markets. They are looking for the technology New Micros has demonstrated the ability to deliver to the market.

The president of New Micros, Randy Dumse, has been with the Forth community for a long time, having first developed the Rockwell R65F11 Forth microprocessor which was a 6502 core with Forth in ROM, followed by New Micros own F68HC11 with Max-FORTH in ROM on a Motorola 68HC11 core. His company carries a full line of single board computers and I/O boards, including both 8 and 16 bit microprocessors.

New Micros is geared to customization. Recently they were working under contract to Crouzet to automate another unnamed company's check printing machine. The specifications seemed simple enough: a driver roller advanced the paper, and an independent print wheel would start and stop to lay the signature plate down on the checks. It had to have 1/12 inch repeatability, since people like checks to be printed accurately and the embossing plate to strike a signature where the signature is expected. However, specifications have no patience for the real world.

Randy and Ron Lovelady, of Crouzet, went through several iterations of optical and magnetic transducers to count pulses from a gear and Randy wrote code to take into account the slack in the rubber drive belt. At one point he developed an adaptive system to compensate for the ever-changing belt tension. Finally, the greatest difficulty was identified.

Two sensors were used on the print shaft, one to indicate a home position, a reliable index point, and, another to resolve a single revolution into 30 counts, 12 degrees each. The print shaft had to accelerate from a dead stop to match the drive shaft speed in only 1/4 a revolution.

This took full power. The motor had to be speed controlled for another 1/2 revolution while the print was laid down. It then had 1/4 a revolution to come to a hard braking stop. Since the drive belt could stretch, the sudden stop caused the wheel to vibrate when settling. If the gear tooth was under the sensor, the count was off. The tooth was being counted multiple times. The problem was resolved by lopping off a gear tooth.

Since there was no point in attempting speed control during the full acceleration, or during full braking, the unnecessary teeth were removed. Only those sensed during speed control were kept.

Eventually, they had a printer which was repeatable, and fast! Total program development time was less than three weeks. The development system for this task consisted of Max-FORTH 3.3 on a F68HC11 single board computer with text files downloaded via Mirror on an IBM PC compatible laptop. Ron chose a custom version of the NMIT-0021 single board computer which includes 8K RAM, 22 parallel lines, 1 synchronous and 1 asynchronous port, and an 8 channel 8 bit A/D. The A/D wasn't required for this project and the final board won't need the off-chip 8K RAM. The RS-232 conversion and power supply circuitry of the larger NMIX-0022 were added, and some custom conditioning put in place on the inputs. Additionally a fully custom interface for the drive motors was hand built for the prototype. It consisted of an "H" bridge driver for each motor and connectors to the micro and the motors.

The interactive Forth debugging cycle offered clear advantages over target compiled C or assembly language. Having a development system on the target identical to the final system also eliminated the effort of switching between development and target systems, as well as an additional step in porting to the final target, and maintaining it. A service diagnostic routine was built in, and a back door provided to allow field modifications. The expected upgrades could be more easily tested with a complete Forth system on chip. An 8K Forth kernel in ROM, either on-chip or off, is an engineer saver!

The **NMIT-0021** is available for the single unit price of **$75.** Quantity discounts are available. Custom versions are available too.

Call **(214)-339-2204** for **New Micros, Inc**. 1601 Chalk Hill Road, Dallas Texas, 75212-5804

Next issue we will present new information from New Micros on their 68HC16 CPU, and later, the RS-422 Multi-Drop Drop Points.

Recruiting

We handle full time and consulting placements. We draw from files that cover the Forth community worldwide. I have met or worked with many of the people in our files through Conferences, publications and consulting. Since I have experience as a programmer/engineer, end-user and a manager, I have a unique set of both technical and human resource skills in the placement field.

We charge agency rates for full time and consultant positions.

Let us put these skills to work for you.

Associates

Each month we will highlight a different Associate. This month, we're bringing to your attention Mr. Gary Bergstrom. You may contract with our Associates either through us, or with them: *there is never a placement charge for an Associate.*

If you are a consulting engineer, programmer or scientist, and would like to learn more about our Associates program, please contact us. We are always looking for good people for our clients.

Contact us at:

Dash, Find & Associates
70 Elmwood Avenue
Rochester, NY 14611
(716)-235-0168 (voice)
(716)-328-6426 (fax)
72050.2111@compuserve.com

Dash, Find & Associates

Technology Brokers

Technology brokering means bringing products and people together in new ways. We specialize in helping clients find these ways, *fast.* Because of this, we specialize in using Forth technology for rapid prototyping, as well as Forth derivatives like Keithley's ASYST™ and Adobe's Postscript™. We provide a range of services from recruiting to consulting to marketing.

My associates can handle software and hardware, digital and analog, programming and technical writing, product specification and project management. We've created joint ventures in Russia. We've been called to give expert witness testimony in a multinational trial.

If you wish to extend an old product, increase staff productivity, bring a new product to market, educate your customers, find new customers or new products: Dash, Find & Associates can help you.

I am Lawrence P. G. Forsley, and I own Dash, Find & Associates.

We have the connections to get things done, fast!

Tech Checks

Technology checks, or *Tech Checks*, give a new perspective on what you are doing right and help you find ways to work *smarter.* We review your goals and direction, and assess your people, program and product. A tech check takes 3 days on site including: interviews with marketing, sales, and engineering staff, as well as reviews of hardware and software, and discussions with users. We observe production, testing and shipping. We give a summary presentation on the last day, and provide a 10 page report.

In one Tech Check we found that a client had developed a multi-tasking operating system working out of bank switched memory in a hand held computer. Unfortunately, the most precious resource was the unswitched global bank, which was largely consumed by the overhead of the multitasker. Consequently, code was moved from the global bank and was duplicated in several banks. This used up data space, and the product began to lose marketshare.

We identified this bottleneck, and devised a method of executing code from bank memory, where a bank context switch occurred with the execution of a polyFORTH word. This new code freed up 64K of bank RAM for data, allowing the client to regain marketshare. Because of the the Forth return stack, programs could be nested nearly without regard for what bank they were in: the calling bank information was always preserved.[1]

Subsequently, over 20,000 hand-held units with our bank select code have been sold, at a value of over $60 million!

Consider using our insights in your work.

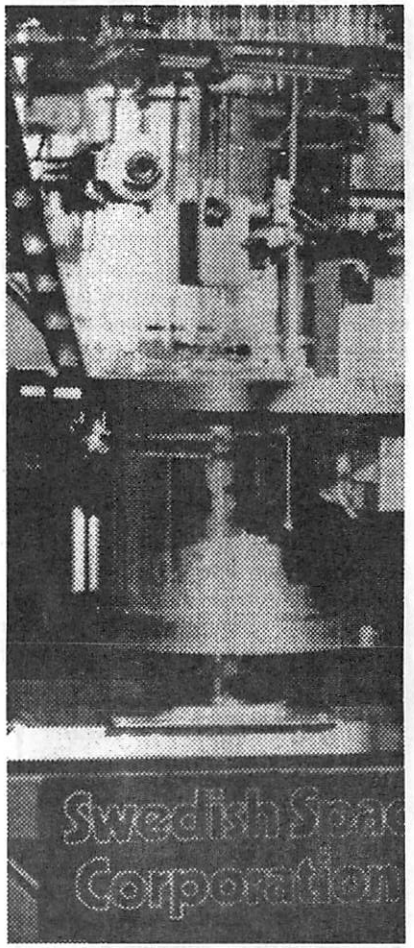
[1] For more information on this project, see "Bank Switched Code for Embedded Systems" in the Proceedings of the 1990 Rochester Forth Conference on Embedded Systems, which is available from the Forth Institute.

Forth in Space

Silicon Composer's 32 bit Forth processor, the SC32, was launched on October 6, 1992 in a Swedish satellite sent into a polar orbit to study the Earth's magnetosphere. The SC32 was developed at the Applied Physics Laboratory at Johns Hopkins University during the past several years for low earth orbit experiments, and then licensed by Silicon Composers as a commercial processor. It was designed into the Swedish satellite, Freja, which was launched by a Chinese Long March II rocket on October 6, 1992 from the Gobi Desert. Rumour has it that President Bush's signature was required for the 32 bit processor export license to China for the launch. Two of the seven experiments in the Swedish Satellite had Forth processors: a German instrument used a Harris RTX processor and the US magnetometer used the SC32.

The Forth processor was a standard SC32 capable of running with a 10 Mhz clock, but reduced to 4 Mhz to allow the use of slower memory with lower power requirements. It was configured with 256 Kbytes of EEPROM and 256 Kbytes of RAM, all addressed as 32 bit words. There was one 16 bit A/D with a 50 usec conversion, and one serial channel for downlink telemetry. Engineers Ben Ballard, Bob Henshaw, John Hayes, David Lohr, Robert Williams and Mary Wong designed the hardware and wrote the software.

The magnetometer has sensors on 3 axes, and measures the Earth's magnetic field. Each axis is sampled at 256 hz with a low pass filter for a total of 3 DC signals and 3 AC signals. A special Z FFT channel is sampled at 512 hz. A hardware digital filter operates at 128 hz and a digital software filter operates at 64 hz to remove additional aliasing when running at the slow telemetry rate. A 512 point FFT is performed once per second. All of this data had to be downloaded



Swedish Satellite, Freja

over a link whose data rate was either 28 kbits/second or 14 kbits/second, which corresponded to either 256 or 128 samples/second. A background task was written to count the number of telemetry interrupts/second to determine the data rate, and reconfigure the hardware accordingly.

The overall data collection rate was 2816 16 bit samples/second (45,000 bits/second), or 24 megabytes/day. The telemetry downlink to a ground station in Northern Sweden was limited at times to a third of this value, so the Forth flight software computed the FFT and could add spectra, which were sent on average every 2-3 seconds.

The satellite has been continuously taking data since October. It was launched during a solar storm, and early scientific data indicates that it could act as an early warning system for solar flares. This is of great interest to power companies in Canada and Sweden, since their electrical grids can be disrupted by solar storms. *An owner of homing pigeons asked to be notified when the satellite detected flares, since his pigeons depend upon the Earth's magnetic field to orient themselves.*

When asked about engineering aspects of the mission, John Hayes noted:

There were more bit flips in the memory than we anticipated. It seems to be over the South Atlantic Anomaly and over the Poles which are high radiation regions. We have this watchdog timer the software tries to tickle, and otherwise the watchdog says to reboot. There were many occasions where a bit was flipped in the program and the software produced strange results instead of crashing. One time the data showed a rectified sine wave [where the negative portion of the signal was clipped] *rather than a sine wave, and the watchdog hadn't rebooted the software."*

John noted that one branch instruction had a single bit flipped which changed the sense of the branch, and this resulted in the truncated sine wave.

The current solution is to reboot the RAM once every orbit, or every 90 minutes, to reduce accumulated program memory errors. John's long term solution is to compute a Hamming error detection code on 16K of program RAM once/second and correct single bit memory errors. There is sufficient bandwidth in the SC32 to allow its other data collection and signal processing tasks in addition to memory testing.

One key to this mission's success is that the Forth processor can be easily reprogrammed from the ground. The initial boot is from EEPROM into RAM, with the EEPROM organized into a simple file system with three copies of the flight program and three copies of an intermediate loader. A new system can be cross-compiled on the ground and a new binary can be uplinked to the satellite. Unfortunately, John notes:

The uplink rate is much lower than expected. It takes 15 minutes to uplink the 16k by 32 bits, whereas we expected to do this in just a couple minutes (about 10 times slower).

This is one of several instances where Forth's re-programmability has allowed space missions to succeed whereas otherwise they would have failed. In addition, the compact Forth code and its efficient, sometimes multiple instruction/cycle execution resulted in one of the highest processing power to electrical power consumed computers. Processing power in orbit will become increasingly important as data collection routinely exceeds telemetry bandwidth.

Now having proven itself in space, as well as on the ground, the SC32 is poised for a significant role in many more space missions.

- by L. P. G. Forsley

For more information contact:

Silicon Composers
655 W. Evelyn Ave. #7
Mnt View, CA 94041
(415) 961-8778

Next Issue: the FRISC-4 Processor