TUG Lines 53

Communications of the Programmer Resource Organization **March/April 1993**

Perspective

The principal mark of genius is not perfection but originality, the opening of new frontiers.

- Arthur Koestler

When it was first introduced, the shareware marketing concept promised to be a panacea for the one-programmer development shop; a real opportunity for the truly "little guy" (or gal) to compete for sales with large developers.

It's been about 10 years since the first really successful shareware products hit the market. In this Perspective, I'll consider, from the author's viewpoint, where shareware marketing has gone during the last decade and where it appears to be heading. This consideration might affect whether you decide to develop or market a product using shareware concepts.

An Overview

From the outset, shareware has promised a number of significant benefits to anyone with an entrepreneurial spirit and limited funding:

Low investment - By uploading a product to a few key Bulletin Boards and then relying on users to download it and share it with others (frequently by uploading it to other boards), one of the biggest costs of product development - marketing - could be reduced to nearly zero.

Low risk - The majority of the remaining risk is the personal time invested by the author in writing the program and its documentation. Couple that with a minimal investment in marketing, and about the only thing you have at risk is your own time. It doesn't get much better than that.

High degree of independence - Without the requirement for large sums of capital, there is little need for investors. And without investors, you're pretty free to do as you wish, and on your own schedule.

Control of the product - Along with the independence comes the ability to determine for yourself exactly what a product should look like and what it should do. Who else do you know who has that kind of freedom? And as the saying goes, "build a better mouse trap, and the world will beat a path to your door".

The Rise of Shareware

The first recorded shareware mega-success authors were Jim Button (PC-File) and Bob Wallace (PC-Write). From what I hear, both eventually became millionaires - and deservedly so. Watching their success, many others quickly began to write applications that were spread from BBS to BBS, across the nation and the world. The growth rate was staggering.

Following the Button and Wallace examples, authors began to turn out full-blown applications which rivaled (and sometimes exceeded) their commercial counterparts developed at considerable expense. The registration fees asked by shareware authors were frequently a fraction of the retail price of equivalent commercially-marketed products. It was a nearly perfect world.

But into that world came a ragtag group of authors who were attracted to the shareware concept only for the money they thought could be made from it. Rather than "doing it right" as Wallace and Button had done, they flooded the world with shoddy software, expecting to make a fast buck. They didn't, of course - but they managed to give shareware a bad name, with highly

Continued on page 2

In This Issue

About TUG Lines . . .

TUG Lines is published bi-monthly by TUG/Pro, as a benefit of TUG/Pro membership. TUG Lines is not available on newsstands or by subscription independent of membership.

Address/Phone

TUG/Pro
PO Box 1510
Poulsbo, WA 98370
Voice:.... 206/779-9508
FAX:..... 206/779-8311

Editorial Staff

Technical Staff

Member Services

Contributors

Membership. Membership in TUG/Pro is $75.00 per year in the United States; $85.00 per year in Canada and Mexico; and $99.00 per year elsewhere. All monies must be in US dollars. Each year's membership includes a one-year subscription (six issues) to TUG Lines, six newsletter disks, and discounts on TUG/Pro products. For more information, request a copy of our current membership prospectus.

Renewals. The renewal information shown on your TUG Lines label tells you the number of the last issue of TUG Lines you are scheduled to receive. If you don't want to let your membership lapse, be sure to renew right after you receive that issue. Of course, we will send you a reminder.

Advertising. We accept a limited amount of advertising, and our rates are very reasonable. For more information, or for our latest advertising rate sheet, contact Carol Taylor at 206/779-9508.

Articles and Submissions. You are encouraged to share your knowledge and experience by contributing articles and reviews for possible publication in TUG Lines. Write for a free copy of our author's guide.

Mailing List. Our membership list is intended for TUG/Pro business only, and it will not be rented, sold or made available to another party or used for any other purpose.

Continued from Page 1

crippled or unreliable products, multitudes of "beg screens", and even threats to those who wouldn't register. I have seen shareware that was pushed out the door so fast that the author demanded payment, but forgot to include either the amount or an address to send it!

The acknowledged registration rate for shareware products became 10%. That means that out of every ten people who were regularly using a shareware product, only one was paying for it. That's the same estimate Ashton-Tate came up with for its commercially-marketed dBASE II. Shareware had "arrived". But it was heading in the wrong direction.

Emergence of the ASP

It was time to do something, and Jim Button stepped into the fray. With the help of some other shareware authors, Jim established the Association of Shareware Professionals, an organization dedicated to setting quality standards for the shareware "industry". Through their efforts, the ASP established a tough set of standards for its member authors which eliminated crippling of software, and provided an ombudsman to help settle disputes between authors and their customers. A serious effort was made to make the general public aware that shareware and freeware were not the same thing. And the ASP established and provided to its members several resources (such as a mailing list of editors of computer-related publications) to help them in their marketing efforts.

There were always differences of opinion between authors and ASP management. But Jim Button's credentials were impeccable (after all, he'd proven to everyone how it could be done), and he prevailed. But when Jim decided to take PC-File into the commercial marketing channels, he decided it would be best if he stepped down from the ASP board. And when he left, so did a piece of his vision.

Without a Jim Button to pilot the ship, dissension in the ranks grew, and by 1992 at least two rival organizations had been formed to serve the shareware author community. These groups (STAR, formed by Scott Miller and Diana Gruber, and ASAD, organized by B. Lee Williams) are now in the process of deciding what their standards will be for authors and their products.

Today's Shareware Issues

There are several events taking place right now which materially affect shareware authors.

You may have noticed the increased number of shareware stores and mail order shareware houses. My mailbox is continually flooded with shareware catalogs, and I even see shareware racks in just about every airport. The stores have become known to the shareware

Continued on Page 3

Continued from Page 2

community as *rack vendors*, and the community is not very happy with them. Basically, rack vendors sell shareware, frequently not making clear to customers what shareware is, and that the products they are getting are unregistered copies. When the customers get the product home, they find out they are supposed to pay a registration fee. And although the fee is normally reasonable when compared to the price of a commercial program, it's typically several times what they just paid for the disk - and in their mind, the product. The reaction is usually to call the store to complain; when that has no effect, they give the author one of those dreaded "How dare you do this?!" calls. Dave Snyder of MVP Software reports that he receives less than one registration for every 2,000 disks shipped by rack vendors. One author (Mike Prestwich of Imagisoft) reports that the recent sale of 12,000 disks containing his Chinese Checkers program brought him only 3 registrations - that means only 0.03% of the people who bought the disk registered it.

According to a long-time ASP member (and current member of the board), shareware sales attributed to rack vendors amount to only 4% of the registrations received by authors - *but they account for 80% of the complaints they get.* Some people are pushing to receive royalties from rack vendors for each copy sold, and some distributors have started to pay royalties voluntarily. But one author I know tells me his total quarterly royalty income for one of his products was less than $50. Several shareware authors are requesting their works be removed from rack vendors' distribution lists.

A second development having a significant impact on the shareware author community is the staggering growth of releases of shareware collections in CD-ROM format. It's hard enough to convince someone to pay you for your program once he's using it. When consumers purchase a few shareware disks from a catalog or store and take them home to evaluate them, programs get some individual attention. But imagine what it's like when your program is bundled with 25,000 other shareware programs - you might be lucky if the consumer even *sees* your program!

While writing this article I talked with one consumer who has recently purchased one of the monster CD-ROM collections. He told me he is using it to choose programs, but in a very unusual way: He reads all the documentation and runs all the programs in the categories, making a list of the various features each offers. Armed with the complete list of features, he chooses those he likes best and goes shopping - for *commercial* software!

It's not too surprising that under these conditions, shareware authors have begun a campaign to stop distribution of their programs on CD-ROM. I recently received a letter from a group of authors who complain that CD-ROM sales are devastating their income, and they're demanding royalty payments from CD-ROM publishers. The supreme irony is that now that shareware authors are finally getting what they sought in the first place - extremely high volume distribution of their work at virtually no cost - it's killing their business.

The final factor affecting shareware marketing today is the narrowing of the niche that shareware has traditionally occupied. Large software developers like Borland, Lotus, WordPerfect and others have begun to develop lower-cost "lite" versions of their products which are aimed squarely at traditional shareware niches. Some of the programs are being priced at the same levels as their shareware counterparts. This makes it tougher and tougher for the shareware author to compete. And the quality of really low-cost, mass-market software (priced at $10 and under) is getting better every day.

Is Anyone Making Money in Shareware?

A friend of mine has deep roots in the shareware community. Last summer I challenged him to name 10 people who are making a significant income from sales of their shareware. The only stipulations I made were that their income had to come purely from shareware (no commercial sales, and no other employment), and that the authors had to be providing at least half of their family's income.

He couldn't do it. In fact, he came up with less than half a dozen names. It was interesting to note that several people he named who have become truly successful in shareware have transitioned over to commercial sales.

It's certainly true that there is an abundance of shareware marketing failures. Have you ever tried to register a program with the address given, only to have your letter returned "Addressee Unknown?" On several occasions I've tried to contact a shareware author on CompuServe, and never received a reply. This is to be expected; every business community has its failures. Unfortunately, many shareware authors think they're going to get rich quick - and when they quickly realize people aren't beating a path to their door to hand them checks, they quickly fold their tents. Marketing your product - be it commercially or via shareware methods - really does take work on your part.

I suspect the majority of profit in the shareware business is being made by *vendors*, not authors. Many times these vendors don't know much about computers or software - they just know how to generate a profit. Unfortunately, many times they're making those profits by exploiting the works of talented people who aren't getting properly paid for their efforts. In the situation mentioned above, Mike Prestwich received 3 registrations for a total gross income of $50.85. At an average retail price of $5.00 per disk, retailers received a gross of $60,000. That is indeed gross.

The Future

Can the average person write a program and get rich by marketing it as shareware?

I don't think so. First, I think you have to be way above average. You have to put at least as much work in marketing your product - before *and* after you write it - as you do in creating the product itself. You can't ever fall in love with your product and assume everyone else will love it too. Instead, you must find out what people want - and what they will actually pay money for.

The competition is fierce, and you have to consider every aspect of your product - how well it performs its assigned tasks, how easy it is for someone to use your product, and how good it looks. As Rosemary West, a consultant and the author of several successful shareware programs says: "People who aren't making money in shareware don't know how to market. They write functional, ugly software." If you're not especially good at graphic design or color combinations, hire someone else to design your screen layout. If you can't write well, find someone who can.

The truth be known, your chances of getting "rich" by marketing shareware are pretty slim. Consider how few successes there have been, and remember - their road was a lot smoother than yours is today.

But even if you don't get rich, you can earn a reasonable income if you're willing to work both hard and smart. *Work hard* to find those niches where there is a market big enough to make it worthwhile for you to create a product - but not enough market to make it worthwhile for the "big boys" to step in. *Work smart* to develop markets, not just products. With the shipment of the first copy of your product, start getting acquainted with your customers. Find out what makes them tick. Discover their wants and desires, and turn that knowledge into new products you can sell to those people you already know and who trust you.

But most of all - no matter how hard you work, and no matter how successful you become - always remember to *invest* time to "smell the roses." You're worth it.

Next time, I'll talk about a book which presents concepts that have changed forever the way I look at business.

Don Taylor

Library Notes

Our thanks to Access Software for providing prizes for both the Issue 52 and Issue 53 product drawings. Elsewhere in the issue you'll find the name of the Issue 52 drawing winner and a description of the prize to be given away in this issue's drawing.

Updates

F-Prot is now at version 2.08A, and it includes additional viruses in its list for detection and eradication. During the past month the news groups on Internet have been covering all of the virus protection programs. Oddly enough, the highly advertised utilities are among the least effective. The effectiveness of each of the virus utilities is evaluated by a group of highly qualified (and objective) experts. One of the comments that has consistently come up is that Microsoft AntiVirus (MSAV), which is provided as part of DOS 6.0, is lacking the ability to detect some of the more common new viruses. And guess which programs consistently rank #1 and #2 on this worldwide list? It's F-Prot and Dr. Solomon's Toolkit. We've been telling you all along: you can't beat the protection F-Prot gives you - at any price! *Stock number UT-MIS-DOS-008*.

New Disks

If you've ever found yourself thrashing through the Turbo Pascal or Borland Pascal manuals, searching for the name or value of a constant, or trying to remember the name of a procedure or function, you're going to appreciate the **Turbo Pascal Reference Book** by Ed Mitchell. Ed is the author of the Borland Pascal Developer's Guide (Que Books), and he was frustrated because Que couldn't print everything he wanted in the book. So he's published the entire reference section as freeware. The result is an incredibly complete reference that covers TP 6.0, TP 7.0, BP 7.0 and Turbo Vision 2.0 - from the operation and use of the IDE right down to every Turbo Vision object. The book consists of a set of ASCII files that make up 10 chapters in a total of 903,222 bytes. If you printed it all out, you would have about 400 pages of single-spaced text for Pascal, plus another 150+ pages for the Turbo Vision material. It's much easier to read than Borland's manuals - and it's all machine-readable, so you can read it with a utility like List, and you can even search for the topic you're interested in. It's a magnificent job. *Stock number RS-PAS-DOS-100-I5*.

Ever wish you could squeeze more performance from your programs, without lifting a finger to do it? Well, now you can - with Norbert Juffa's **Optimized Runtime Library Update** (version 1.2) for Turbo Pascal 7.0 and Borland Pascal 7.0. He has developed complete shareware runtime replacement libraries for real mode, protected mode and Windows versions. Compile your programs using these libraries, and you can not only speed up your program execution - you can increase the accuracy of your calculations as well. The overall improvement in arithmetic routines is almost 30%, with a resulting increase in accuracy in transcendental functions. String operations

average 7% faster, and random number generation using Reals ranges from 10-20% faster. Also, the UpCase function has been extended to properly upper-case the international characters. And while he was at it, Norbert fixed several bugs in Borland's RTL. Much of the code is provided in source form (including the code for the String and Math units, which Borland doesn't give you in their RTL package). A total of 749, 492 bytes of incredibly powerful software and documentation. *Stock number PT-PAS-MIS-015-I5.*

We've been looking for a really top-notch source of information for Pascal and C++ Windows programmers, and we've discovered something that has exceeded our wildest hopes. It's called **Windows Programmer's Journal**, and it's a magazine published exclusively on disk. It's extremely user-friendly, and caters to everyone from Windows beginners to experienced hackers. It's currently being published in two formats - straight ASCII text and as a Help file you can view from the Windows Help system. Here's a summary of what one issue (#3) contains: A beginner section that teaches how to design dialog boxes and insert controls; A tutorial on how to create "designer" buttons with bitmaps for labels; Using file version stamping to be able to recognize special files; Painting in Windows; Advanced C++ and Windows; Programming Windows with Borland Pascal; and tips and tricks. Source code from examples is included on disk. We've grabbed the first 5 complete issues and crammed them on this disk for you. If you're interested in programming in Windows, this will give you a great boost. *Stock number RS-MIS-WIN-200-I3.*

If you sent in a registration card for any Borland programming product, you have probably received copies of Borland Language Express. The magazines are great, but they don't come with source code. Now you can get the source code - in Pascal and C++ - from the first 5 issues, on the **Borland Language Express Code Disk**. On this disk you'll find it all - source, executables, everything - in C++, Pascal, assembly and Paradox Engine files. A total of 131 files and 574,884 bytes. *Stock number RS-MIS-MIS-005-I5.*

No doubt about it: installing your Windows program is an irksome task, at best. But we've discovered a neat piece of shareware that makes creating installation routines a snap. It's called **EDI Install Pro**, a package that puts a very powerful and extensible (but very easy to use) installation capability at your fingertips. Utilizing a simple script bearing a staggering resemblance to an .INI file, you can create an installation package that allows for custom and partial installations, .INI file modifications, distribution file compression, and disk space management, to name a few of its many features. In as fierce a market as Windows applications, the first glimpse a customer gets

of your package is crucial, and this installation program makes a great first impression. (In fact, it just happens to look a lot like most of Microsoft's applications installation programs..) This disk also contains the **Windows Enhanced Dialog Library**. WEDL is a shareware package that gives your dialogs formatted entry fields and entry field validation, as well as automatic variable update, 3-level undo, case conversion, overtype mode, and extended field editing. Drag and Drop is also supported. Very nice. *Stock number PT-MIS-WIN-012-I5.*

Ever wonder what "Visual Pascal for DOS" would be like? How would you like to be able to design dialog boxes, popping buttons, check boxes, radio buttons, input lines or whatever into place in your box, then guide them in place with your mouse? Well, with David Baldwin's **Dialog Designer for Pascal**, you can do all that - and a whole lot more. We've used the commercial dialog designer for Turbo Vision, and frankly, this is better. It is for Turbo Vision what the Borland Resource Workshop is for Windows. You can quickly lay out any dialog box using any of the standard controls, and in the process you can change just about any aspect of any control. Once you have designed your dialog, you can immediately test its operation - without leaving Dialog Designer. When you have the design just right, you can save it, and then you have 3 options: (1) You can save it as a Pascal resource in a resource file; (2) You can generate source code and display the source on the screen; or (3) You can generate source and save it in a file which can be included in your program. The source code option generates a function which will create the object you designed and pass you a pointer to it. The operation of Dialog Design is very flexible, and enables you to generate source for virtually any descendant of a TDialog object. Two versions are included on the disk: Version 2.2 is for Turbo Pascal 6.0, and Version 3.01 is for TP 7.0/BP 7.0. If you're programming with Turbo Vision (or even *thinking* about it) You Gotta Have This Program! *Stock number TV-PAS-DOS-002-I5.*

Looking for a terminal package that can enable your Pascal program to emulate a versatile Avatar terminal? The **Turbo Pascal Avatar Level 1 Console Kit** (PAvatar) is a shareware toolkit from Gregory P. Smith which enables you to add an Avatar or ANSI terminal to your program. It emulates TTY, ANSI, AVT/0+ (extended level 0+ with ANSI fallback), AVT/1 (full level 1) and standard VT-52 terminals. Pavatar is also multitasking aware. *Stock number PT-PAS-DOS-013-I5.*

WinThreads is a non-preemptive multi-threading shareware library, implemented as a Windows DLL. **NewTrak** is a freeware C++ tool that is an extension to C++'s own memory management routines. It validates deletes, detects memory overruns and underruns, and it identifies memory leaks. (For a detailed description of this disk, see "The C Scape" in this issue.) Stock number PT-CPP-WIN-014-I5.

Turbo Vision Disk 3 contains two useful packages for C++ programmers. **Dialog Designer for C++** contains most of the capabilities of the Pascal version (see above), with the exception of being able to save dialogs directly to resource files. The other package is the **Turbo Vision Resource Workshop**, a shareware utility that emulates the Borland Resource Workshop to design dialog boxes for TV C++. *Stock Number TV-CPP-DOS-003-I5.*

Turbo Vision Disk 4 contains four utilities for C++ development. **TVTools** is a collection of Turbo Vision tools; **MDIDLG** is a set of OWL routines that turn dialog box definitions into MDI-compatible windows; **QDHelp** is a "quick and dirty" Help file maker; and **DBPutS** is a debugging tool handy for debugging DDE clients and servers. (For a more complete description, see "The C Scape" in this issue.) Stock number TV-CPP-MIS-004-I5.

If you've been reading Tim Gentry's columns, you know he's a real fan of the Internet. Have you ever wondered just what Internet is, and how you can use it? Well, for about 25 bucks you can go out and purchase a copy of **Zen and the Art of the Internet**, "the" reference book on the subject. Or, you can get the pre-release version of the book on this disk, courtesy of The Gutenberg Project (which publishes freeware books on Internet). The entire book is on this disk. Plus, you get **FixUtil** (version 4), a collection of routines that will help any programmer, regardless of programming language. And there's **DRPeek** (short for Data Record Peek), Jerry George's utility for analyzing fixed-length data record files. With this utility you can figure out what the data structure is, and output the structure, so you can quickly write routines to manipulate the data. *Stock number RS-MIS-MIS-101-I5.*

We wanted to pick the best Windows shareware communication program, and we selected **MicroLink**. It's really an elegant program that provides the flexibility you need, with built-in convenience. We liked the layout and the high degree of customization we got, without being faced with screenfuls of menus and data entry screens. It also has one intolerably cute feature: a simulated LED display bar across the top of the screen; the LEDs "light" to show the status of TX/RX, CD and so forth. *Stock number CO-MIS-WIN-001-I5.*

When we went searching for the best DOS-based shareware communications program, we settled on **Telix** (version 3.21) from deltaComm Development. It's a classic - it's been around for several years - and it just keeps getting better. Includes a powerful scripting language capability. *Stock number CO-MIS-DOS-002-I5.*

Want to be able to easily create documentation for your project? **Envision Publisher** is a nice DOS-based shareware application that can turn out good-looking documentation with ease. *Stock number BF-MIS-DOS-002-I5 (a 2-disk set).*

When Ordering Disks

Use the order form that accompanies this issue. Please note that stock numbers ending with "I5" are in 5.25", 360K format. Stock numbers ending with "I3" are 3.5", 1.44M format.

Pascal

Top of the Heap

Don Taylor

Those who attended GeTUGether last summer heard Zack Urlocker, Borland's Pascal product manager make a mysterious pronouncement: that Borland was working on a major commercial application for the mass market, and that it would be written in Pascal, not in C++.

I'll admit that more than once it crossed my mind that this was nothing more than a bit of marketing hype. But this was not just anybody - this was Zack, someone I have come to respect and trust. Still, the months rolled on, and no major application in sight.

Then the news broke, on the front page of the May 31, 1993 issue of PC Week. The mysterious product is none other than version 5.0 of Quattro Pro for DOS!

The new version is supposed to ship this fall, and it will incorporate several new features including a look that bears a resemblance to the Windows version of Quattro. But the more important question is, "Why Pascal?"

So far, the answer is the ability to shrink the size of the EXE, and to improve its speed performance. Let's think about that a moment. This is no Public Relations maneuver to convince us that Borland is committed to supporting Pascal. They have taken perhaps the most profitable product they have, which competes for sales in the toughest market Borland is involved in, and they have relied on Borland Pascal - not Borland C++ - to give them the best edge they can find.

Those of us who still program in Pascal can take heart in that. Thanks, Borland.

◆ ◆ ◆

In this issue you'll find three articles I think you'll benefit from. The first is Bob Crawford's next installment in his series on image processing. This time, Bob demonstrates how to manipulate Targa files to brighten, darken, invert, split colors and convert to grayscale.

Then Paul Gallagher concludes his mini-series on state programming and object morphing, including some code you can use to morph your own objects (gee, sounds kinda kinky, doesn't it?).

Finally, you'll find my article on using streams, without having to use Turbo Vision. Well, almost...

Image Processing: Point By Point

Bob Crawford
Bowling Green, Kentucky

This outing we are going to look at a few image processing facilities which are collectively known as point processes. The reason for the name is quite straightforward - they all work by examining the pixels of the input image one-by-one, in isolation. Based solely on the value of a given pixel, the value (presumably different, if you want the result to be interesting) of the corresponding output pixel is computed.

Since such processes do not use any contextual information such as the values of nearby pixels, they are somewhat limited in scope. Nevertheless, a good deal of damage can be done with point processes and they do have the advantage of being fast in comparison with more sophisticated manipulations. We will use them for brightening and darkening images, inverting images, separating color images into their red, green, and blue components, and turning color images into grayscale images.

To the Point

Up to now we have concentrated exclusively on true-color (24 bit) Targa format files, but several of the things we are going to bump into deal with grayscale images or close relatives. You can store grayscale images in the true-color format (just use the same value for all three components of a color), but it's a terrible waste of space. Instead of storing three copies of the same value, it makes sense to store just one.

This is how the Targa type 3 storage format works. It has the usual Targa file header (this time with a 3 in the ImageType slot), followed by a single byte for each pixel in the image, storing the gray level of that pixel. In our 320 x 200 case, this amounts to 64,000 bytes of image information plus the 18 bytes for the header - nearly a two-thirds savings from the 192,018 bytes required for a 320 x 200 true-color bitmap. (Note that such a file does not contain any palette information at all. Except for the 3 in the header, it does not "know" that it is a grayscale file. Indeed, it could be displayed using any palette you like, although most would make for strange results.)

To simplify working with these files, I have added a routine:

```
WriteTargaGSHeader(var f : file;
  Width, Height : word);
```

to the `TargaU` unit. It works just like `WriteStandardTargaHeader`. You feed it an untyped file (already opened using `ReWrite(f, 1)`) and the width and height of the image, and it writes the needed eader to the file.

These grayscale files arise, naturally, as the output from a request to turn a true-color image into a grayscale image. They can also be used for the output from a request to separate the RGB values into different files, and that's exactly what our point processing program will do. To enable you to view all these kinds of files, I have included a program called TGSView on this issue's Newsletter Disk. It takes the name of the file to be viewed as its first command line argument. If GSFish is a grayscale picture of a fish, for example, calling

```
TGSView GSFish
```

will display said grayscale fish (and wait patiently for you to press a key).

If, on the other hand, you start with a true-color picture of a fish and ask for it to be separated into its RGB components, the result will be three files (say RedFish, GrnFish, and BlueFish). Each of these files has the same structure as a grayscale picture, and each could be displayed (in gray) just like GSFish was above. TGSView takes an optional third parameter to allow you to view these files in the colors they were intended to have. The third parameter must be either r, g, or b, depending on whether you want the image displayed with a red, green, or blue palette. Thus to view the RedFish file (in red), you type

```
TGSView RedFish r
```

By the way, I have also included on the Newsletter Disk an extremely simple program that will give you (in text format) information about a Targa file - TGAInfo. Just give it the name of the Targa file on the command line and it will read the header and display the information it contains in a reasonably readable fashion.

More to the Point

Now on to the techniques themselves. One nice thing about point processes is that they all operate in an essentially similar fashion. The main program here, Point, exploits this similarity by setting up a single routine (`PointProcess`) to manage the overall flow control. `PointProcess` is passed a procedural parameter (based on what is requested on the command line) which dictates the particular process to be employed. The general form for using the program is:

```
Point <InputFile> <OutputFile> <Command>
```

Here Command can be b (brighten / darken), i (invert), s (separate colors), or g (convert to grayscale). In one case (brightening / darkening), these three arguments need to be followed by additional arguments. We now turn to looking at the individual manipulations.

Brighten / Darken

Brightening a picture simply amounts to adding a given increment to each of the RGB values for every pixel, making sure the results do not go above 255 in any component. Although the usual application would use the same increment for each color, Point allows you to specify the increments individually. The increments are specified (in the order RedIncrement, GreenIncrement, BlueIncrement) at the end of the command line, so a

typical invocation might look like

```
Point F320x200 BrightF b 30 40 50
```

Of course, if you need to darken a picture in one or more of the colors, just make the corresponding increments negative. Point makes sure the resulting components do not drop below zero.

Invert

Inverting here refers to replacing each color component value x by `255 - x`. When this process is applied to a grayscale image, the result looks like a photographic negative. It's kind of a weird thing to do to color images, but if you are into strange pictures, have at it.

Separate Colors

The idea here is simply to split a color image into three files - one storing the red values, one storing the green values, and one storing the blue values. No actual changes are made in the values themselves. Simplicity itself.

Convert to Grayscale

It is not immediately obvious how one should go about turning colors into shades of gray. For one thing (and it's a biggie), the human optical system responds quite selectively, depending on the color. It responds differently to reds, for instance, than it does to greens and blues. In particular, our eyes are not nearly as sensitive to shades of blue as they are to shades of red and green. In fact, the National Television Standards Committee defines a formula for accommodating for these differences:

```
GrayLevel = 0.299 * Red + 0.587 * Green + 0.144 * Blue
```

Notice that the coefficients here are positive and add to 1, so underflow and overflow will never be a problem. In many cases, low-resolution images actually look better in grayscale than they do in color. The smooth gradations in tone help to eliminate the jaggies.

Summary

You should realize that **Point** was never intended to be a professional-quality program - and it certainly succeeds at that! To mention just a few of its shortcomings, I might start with speed. There isn't any. A little buffering would do wonders here. Then there is the limitation on image size - 320 x 200, that's it. Using protected mode would ease the memory strain caused by SVGA modes and make them reasonably easy to support. Finally, four processes (five if you count brightening and darkening as two separate processes) are not very many. Well, **Point** is only meant to, er, *point* you in the right direction. You can easily add your own commands and processes to point to do anything your imagination can conjure up.

Let go! Experiment!

Next time we will consider processes which consider a neighborhood of nearby pixels when computing the output value for a given pixels - the so-called *group processes.*

***Bob Crawford** is a professor in the Computer Sciences Department at Western Kentucky University, and a contributing editor of TUG Lines on a variety of subjects. He can be reached on CompuServe at 71121,777.*

Object Morphing: State Programming with Turbo Pascal Part 3

Paul Gallagher
Mt. Waverley, Victoria, Australia

In the first two parts of this series, we examined how the OOP paradigm limits the ability to model real-world problems, and we explored several alternatives for turning those ideals into reality. In this final installment we'll see how to actually implement a morphable object - complete with a general morphing unit you can put to use and a demonstration program you can run.

Doing a Morph

Once we have registered our State Objects and set up the State Object Register (SOR), the actual process of morphing an object is a trivial one. The State Class Definition Object (SCDO) could define a `Morph` method like the following:

```
procedure TStateClassDefObject.Morph(newState :
          word);
begin
 { Call a prepare-for-morphing routine }
 StateDone;
 { Change our VMT }
 word(self):=newState;
 { Call the init routine (of the "new" object)}
 StateInit;
end;
```

[I was tempted to call the procedure `"SetState"`, but resisted since (unfortunately) this and other references to "state" tend to conflict with the concept of state as used in Turbo Vision.]

`newState` is actually the VMT offset of the new data type - which we have conveniently stored in the SOR, accessible to all. `Morph` can be called from "outside" the object (by a user of the `Morph` Class), or from other methods of the object. We have succeeded in changing an object's type at runtime in a very efficient manner. Some points to note:

1. No memory allocation, deallocation or copying was involved.
2. All the data in the "old" object is still present in the "new" one.
3. The address of the object has not changed.
4. We are still "type-safe".

`StateDone` and `StateInit` are virtual methods that provide "hooks" for descendant State Objects to do any necessary housekeeping before and after morphing. One of the most important tasks is the management of any dynamically allocated variables defined in descendants of the State Class Definition Object. Because we do not know what object type is going to be current when the instance is finally disposed, de-allocation of these variables cannot be left to the destructor. `StateInit` should allocate and `StateDone` should de-allocate memory for any such variables. Additionally, a default destructor could make a call to `StateDone` instead of including its own de-allocation code. Similarly, a default constructor could use `StateInit` to handle allocation.

`StateInit` is like an alternate `constructor Init`. It should do any initialisation required, with one important difference - many (if not all) of the object's fields may still contain valid data (left over the previous "type"), so they will probably be preserved rather than initialised as you might expect.

Descending Non-State Objects from a State Class

A special note is warranted to consider the case of inheriting a SCDO or associated State Object in order to create a new object that is not intended to be part of the State Class (or a State Object at all, for that matter).

The main task is to ensure there is no chance that an inherited method misguidedly attempts to "self-morph". The easiest way to do this, of course, is simply to override the `Morph` method, recasting it as an empty or perhaps an abstract method for debugging purposes..

A Demonstration Of Object Morphing

The code accompanying this article demonstrates how object morphing can be implemented. (I'm not going to document the example in detail, since it largely just puts into practice the concepts that have been discussed so far.) Figure 1 reveals the object hierarchy for our example.

The `MORPH` unit is constructed as a fairly generic "template" unit which can readily be used to incorporate morphing into any application. `TStateObjectRegister` is intended to play the role of an application-wide State Object Register, so it keys object records on the basis of Class as well as State ID. It is not instantiated in the unit, to prevent conflicts that may arise in more complex applications.

`TStateClassDefinition` is an abstract SCDO that does everything required to implement morphing. To create a State Class, you inherit this object and should only need to add to it your application specific code (forget the mechanics of morphing - just remember you can do it!).

An enhancement which is included (perhaps undesirably) at the abstract level, is that requests to "morph" (or change state) are made on the basis of a unique "State ID" which is application-defined. The SCDO then automatically goes through the process of consulting the SOR to find out what the new object type (alias VMT offset) should be. If the application in fact uses the SOR to find out what the State ID is in the first place, then, well... you're double-handling the data! (This behaviour is easy to pull out - it actually simplifies the SCDO substantially).

The `DemoSC` unit defines a sample State Class, and it includes a procedure which registers all the state objects for morphing. `TData` is the non-abstract descendant of

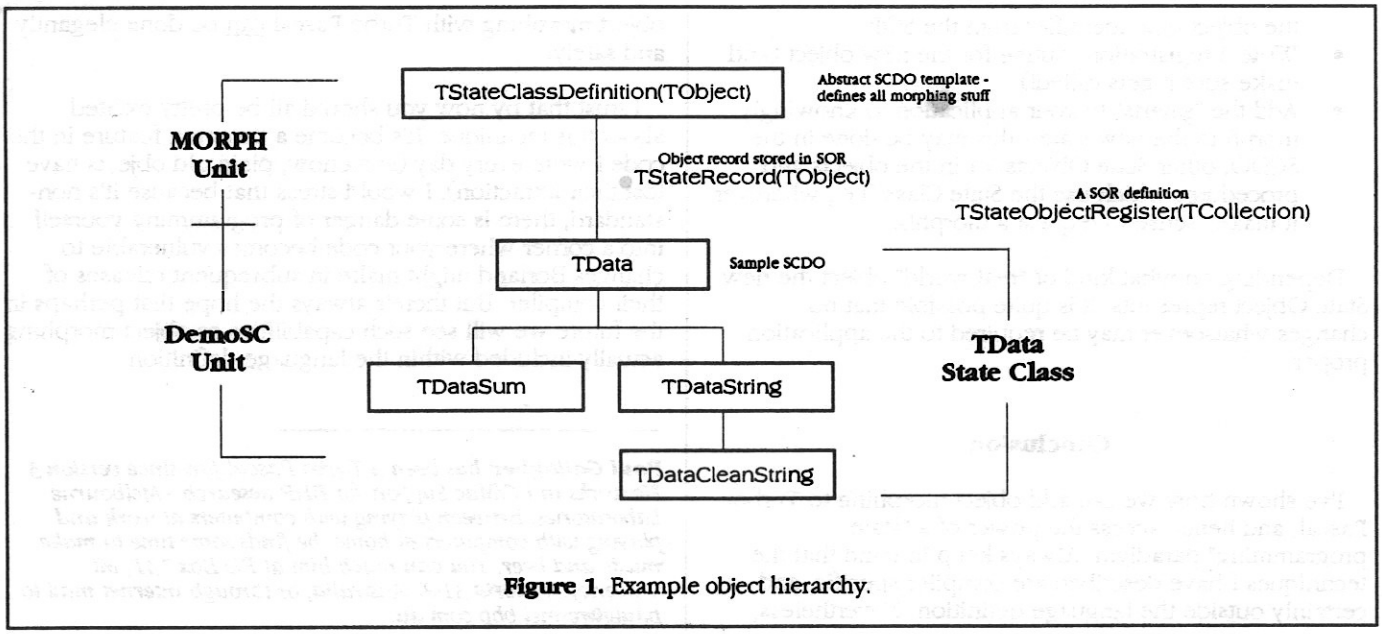

Figure 1. Example object hierarchy.

`TStateClassDefinition`. It is a simple object, containing a small array of bytes (which are initialised to an arbitrary data set) and a single method : `Display`. Because it is the "State Class Definition", the data field and single method "define" what the `TData` state objects are all about. `TData.Display` simply writes the data to the screen as a stream of bytes.

`TDataSum` re-defines the display method: instead of displaying the bytes themselves, it displays their arithmetic sum, demonstrating "self-morphing" by then changing back into the Base State (`TData`) and again calling its display method (which is now `TData.Display`).

`TDataString`, which in its `Display` method interprets the data as a string when writing it to the screen, demonstrates how to handle dynamically allocated variables in a state. In addition to new data fields, it introduces a new method, `TDataString.Convert`, which simply copies the data set into a dynamic string variable. Because it is not in the SCDO, it is thus not part of the class definition, and programs using the State Class should pretend it does not exist. This is, in fact, enforced by declaring the method as `private` (for all you C++ programmers, "private" actually means something more like "protected" in this context).

`TDataCleanString` is actually descended from `TDataString` rather than `TData` (just to show it can be done). It over-rides the convert method, so it also removes non-printing characters (producing a "clean" string).

That's a fair bit of functionality, all-be-it trivial. What really *is* trivial, however, is the manipulation of our State Class once it's been defined:

DEMO.PAS is a quick "test drive" of the `TData` State Class. It first instantiates a `TStateObjectRegister` (from MORPH.PAS), then it calls `RegisterDemoClass` (from DEMOSC.PAS) to make the class morphable. It is then ready to go to work.

This demo program follows the practice I've recommended. It first creates an instance of `TData` (the SCDO) as its "handle" into the State Class. Once it's got the instance, it just "morphs around" a bit, calling the `Display` method to see what happens.

Assessing the Concept of Object Morphing

Object morphing, as we've described here, has succeeded in allowing objects to change type ("state") at will. The owner of the object instance does not necessarily know or care that the morphing is taking place, but it should be emphasised that the owner *can* find out this information, if it wishes to do so.

The implementation has specified requirements for objects that are to be made morphable (e.g., they must be virtual and dynamic), but I think in most cases the potential benefits far outweigh these fairly trivial restrictions.

It is also possible to "retro-fit" morphability into existing object definitions. This may entail some redesign to make the object class conform to the requirements for morphing, and it may require some extra work in determining exactly where the VMT offset field in an object instance is actually located. Not usually major problems.

The true test of the elegance of object morphing is to assess the ease with which a State Class may be extended (perhaps by a third party). Adding a new State Object proves to be a very straightforward process:

- Create a new object descended from the SCDO or another State Object.
- Define a unique key for this State - used to extract

the object type identifier from the SOR.
- Write a registration routine for the new object (and make sure it gets called!).
- Add the "smarts" to your application to knowingly morph to the new state - this may be done in the SCDO, other State Objects, or in the objects and procedures which use the State Class (i.e., wherever it makes sense to request a morph!).

Depending on what kind of "real world" object the new State Object represents, it is quite possible that no changes whatsoever may be required to the application proper.

Conclusion

I've shown how we can add object morphing to Turbo Pascal, and hence access the power of a "state programming" paradigm. Always keep in mind that the techniques I have described are compiler specific, and certainly outside the language definition. Nevertheless, object morphing with Turbo Pascal can be done elegantly and safely.

I trust that by now you should all be pretty excited about this technique. It's become a common feature in the code I write every day (somehow, plain old objects have lost their attraction). I would stress that because it's non-standard, there is some danger of programming yourself into a corner where your code becomes vulnerable to changes Borland might make in subsequent releases of their compiler. But there's always the hope that perhaps in the future we will see such capabilities as object morphing actually included within the language definition.

***Paul Gallagher** has been a Turbo Pascal fan since version 3. He works in PC/Mac Support for BHP Research - Melbourne Laboratories. Between playing with computers at work and playing with computers at home, he finds some time to make music and beer. You can reach him at PO Box 731, Mt. Waverley, Victoria 3149 Australia, or through internet mail to paulg@resmel.bhp.com.au.*

Ex-Stream Prejudice

Don Taylor
Poulsbo, Washington

Much hoopla has been made about streams and their ability to store objects of any type. Basically, a stream is an object which enables you to save or retrieve information to a variety of devices - disk files, buffers or whatever. By their very nature, streams are both flexible and powerful. In fact, streams make it child's play to store the entire environment of an executing Turbo Vision application to disk - desktop, global variables, everything.

Streams were designed with objects in mind. Turbo Pascal doesn't recognize a typed file of objects, and even if it did, virtual methods which are declared in many objects would create havoc with conventional typed file I/O because the virtual method's addresses aren't determined until run time.

Borland's solution to the problem is the `TStream`, an object which implements the stream concept and enables you to save and retrieve objects when using Turbo Vision.

But is that the entire story?

A Case of Stream Prejudice

A `TStream` is actually an abstract object used as a basis for inheriting more specialized stream objects - `TDosStream` is a specialized descendant which reads and writes conventional DOS files; `TBufStream` is a descendant of `TDosStream` which provides buffered file I/O for DOS; and `TEmsStream` is a descendant of `TStream` which lets you read and write streams to and from EMS memory.

All of these objects are contained in Turbo Vision's `Objects` unit. And that's where our story really begins.

Apparently in their zeal to create a stream capability for Turbo Vision, the designers at Borland failed to fully realize that programmers using Turbo Pascal and Borland Pascal might want to use that capability without using Turbo Vision itself. Add to that Borland's prejudice for associating streams exclusively with objects. It tends to make one believe that TP/BP streams can only be used with objects.

Consider this quote from the introduction to streams in the TV 2.0 Programming Guide:

> "A Turbo Vision stream is a collection of *objects* on its way somewhere: typically to a file, EMS, a serial port, or some other device. Streams handle I/O *on the object level rather than the data level*."

To emphasize my point, I've italicized portions of the text in the quote. Now tell me - if you were looking for a streaming capability in TP or BP and you found reference to it only in the Turbo Vision manual, and in that manual you read the statement above, wouldn't you assume that you *had* to use Turbo Vision to use streams, and that streams were *only* for use with objects?

Well, prepare yourself for a couple of earth-shattering revelations:

True Fact #1: You don't have to embrace Turbo Vision to use its stream capabilities. All you have to do is use the Turbo Vision `Objects` unit in your program.

True Fact #2: Although `TStreams` and their descendants are themselves objects, they aren't limited to use with objects. In fact, you can use them to work with any data type!

A Little Perspective

TP and BP provide two powerful file I/O capabilities. At one end, we have typed files which provide high-level file management - random access, fairly sophisticated error detection, and an ease of use. Typed files have one drawback, however: they must comprise a set of data records of identical structure.

On the other end of the scale, we find untyped files and the ability to process data on a binary level using the `BlockRead` and `BlockWrite` procedures. This provides a very fast means of I/O, but it saddles us with the responsibility to juggle a lot of details, typically writing specialized data-handling routines for each application.

In between these two I/O techniques lies the stream. It affords us both random-access capability and error detection, with an additional bonus: it lets us easily mix and match data in files.

You see, a stream is really nothing more than a file that can handle a "laundry list" of data of varying lengths. It was designed that way to handle objects, but it can be used for data of any kind. For both its reads and its writes, a stream has to know the name of the variable to/from which the data will be transferred, and information that will lead to the knowledge of the length of that data.

In the case of objects, that requirement is met by defining and establishing a stream registration record. That record assigns a unique identification number to the object; when the object is to be written, the ID number is written first, followed by the object itself. The information in the record tells the stream object's `Write` procedure exactly how many bytes to write. When the object is to be retrieved, the opposite operation takes place. The ID is read, its corresponding registration record is located, and the correct number of bytes to be read is found in the record.

If we want to use streams to store data items other than objects, we must take on the responsibility for telling the stream exactly how many bytes to read or write for each item. This is a simple and straightforward task, but we must be ever-vigilant as we do so, for streams have no way to check for errors on our part.

A Example by Recollection

In Issue 52 (see "Taking Up a Collection", page 19) I used the program `Wince` as an example of how to use Turbo Vision's `TCollections` apart from TV. I'm going to use the same program to demonstrate the use of streams for both object and non-object data.

The `Wince` program defined a descendant of a `TCollection` called a `tSCollection` to store an unspecified number of records containing the names and datestamps of files in directories. In addition, the program needed to store 3 variables - `WinIdx`, `SysIdx` (both integers) and `SaveDate` (of type `DateTime`) in the same file. Storing this mixture of apples and oranges in a single typed Pascal file would likely result in a fruit salad - but as we'll see, it's a snap for a stream.

We'll take the challenge on in top-down programming fashion.

First, we're going to be storing and loading a new object (a `tSCollection`), so we need to create a registration record for it:

```
const
 rSCollection : TStreamRec = (
  ObjType : 10010;
  VmtLink : Ofs(TypeOf(tSCollection)^);
  Load : @tSCollection.Load;
  Store : @tSCollection.Store
 );
```

Here the `ObjType` value is the unique ID value we've defined for this new object. In the `VMTLink` field we have given the TP/BP system the mechanism for calculating the offset into the object's VMT, so the system can determine the size of the object - this will be used when storing and loading the object. Finally, we're telling the system where to find the routines which actually perform the load and store operations.

Next, we need to tell our program to register the object, so it will be ready when we want to load and store objects of type `tSCollection`. Somewhere in the initialization portion of the program, we include the line

```
RegisterType(rSCollection);
```

Now we need two routines that will save and retrieve all the data from the stream. First the save routine:

```
function DataSaved : Boolean;
{ - Save the status collection to a
     specified file }

var
 ItemStream : PBufStream;
 Status     : Integer;
begin
 ItemStream := New(PBufStream, Init(vExePath +
   cDataFileName, stCreate, 1024));
 Status := ItemStream^.Status;
 if Status = stOK
  then begin
        GetDOSDateTime(vOldColl^.SavedDate);
        ItemStream^.Put(vOldColl);
        Status := ItemStream^.Status;
       end;

 Dispose(ItemStream, Done);
 DataSaved := Status = stOK;
end; { DataSaved }
```

Let's take a look at what we've just done. We've created a PBufStream (a pointer to a buffered DOS stream we've constructed in memory). The stream is given a predefined name, it's created with stCreate, and it is allocated a buffer of 1,024 bytes. If there were an error creating or opening the file, we would not get a value of stOK when checking the stream's Status field. When saving data, Wince updates the SavedDate field in the vOldCollection object. It then sends the entire collection to the stream with the stream's Put method, and then it checks the result to use as the function's return value. The stream's work is finished, so it is disposed of.

Retrieving data is very similar:

```
function DataRetrieved : Boolean;
{ - Retrieves a status collection from a
  previously saved file }

var
 ItemStream : PBufStream;
 Status     : Integer;
begin
 ItemStream := New(PBufStream, Init(cDataFileName,
   stOpenRead, 1024));
 Status := ItemStream^.Status;
 if Status = stOK
  then begin
        vOldColl := pSCollection(ItemStream^.Get);
        Status := ItemStream^.Status;
       end;
 Dispose(ItemStream, Done);
 DataRetrieved := Status = stOK;
end; { DataRetrieved }
```

This time, the stream is opened for reading with stOpenRead, and the collection is retrieved with the stream's Get function.

When we call the tSCollection's Put and Get methods, we automatically put into motion a mechanism which is part of the TP/BP stream-handling system. The Put procedure locates the registration record for the object, then it writes the object's ID to the stream, and finally it calls the object's Store method. We are free to tell Store just exactly how we want to send our data to the stream:

```
procedure tSCollection.Store(var S : TStream);
begin
 S.Write(WinIdx, SizeOf(WinIdx));
 S.Write(SysIdx, SizeOf(SysIdx));
 S.Write(SavedDate, SizeOf(SavedDate));
 TCollection.Store(S);
end; { Store }
```

To keep some semblance of sanity, we've written the 3 variables (which will be written to the stream each time) first, followed by the variable-length collection. We have used the TBufStream's Write method to write the variables; note that the TCollection.Store method already knows how to write the collection to disk.

The tSCollection's Get function performs the complement of the Put procedure. It calls the object's Load constructor, which we write as:

```
constructor tSCollection.Load(var S : TStream);
begin
 S.Read(WinIdx, SizeOf(WinIdx));
 S.Read(SysIdx, SizeOf(SysIdx));
 S.Read(SavedDate, SizeOf(SavedDate));
 TCollection.Load(S);
end; { Load }
```

As you can see, this is truly the complement of the Store method we wrote earlier. And it is here that we must exercise utmost caution - the data loaded must exactly match that stored. If there are too few or too many items, or if the sizes of the items don't match, you'll get a stream error. *Caveat programmer.*

Special TCollection Requirements

In the previous article, I told you that TCollection objects can manage any type of data - and that's true. Because of that ability, a TCollection has to be told how to put and get each of the items it manages to or from a stream. This is accomplished by overriding the TCollection descendant's PutItem and GetItem methods:

```
procedure tSCollection.PutItem(var S : TStream;
  Item : Pointer);
{ - Store one pFStatusRec item to the stream }
begin
 S.Write(Item^, SizeOf(tFStatusRec));
end; { PutItem }

{--------------------}

function tSCollection.GetItem(var S : TStream) :
  Pointer;
{ - Load one pFStatusRec item from the stream }
var
 Item : pFStatusRec;
begin
 New(Item);
 S.Read(Item^, SizeOf(tFStatusRec));
 GetItem := Item
end; { GetItem }
```

That's it - each time tSCollection.Store or tSCollection.Load is called, it will, in effect, call the tSCollection's ForEach procedure in concert with PutItem or GetItem to process each of the data items in the collection.

Jumping Naked into the Stream

Until now, we've been talking about storing and loading objects in combination with other data types. I'll be the first to admit that due to the number of requirements to be met (including some esoteric ones, like stream registration), this is not exactly a simple and straightforward process.

What if you want to make use of streams without the ability to store and load objects? What if you want to forsake Turbo Vision altogether - to cast it outright into the Pit of Eternal Parentheses, using its streams only to write simple (non-object) data files?

I'm glad you asked that question (you *did* ask it, didn't you?).

You see, there's a way to cut through all of the complex mechanisms which enable streams to work with objects. And when you take that path, things become greatly simplified.

On this issue's Newsletter Disk, you'll find a file named STRMDEMO.PAS. This demo program shows how easy it can be to use streams as data files in any Pascal program. The demo program's `SaveData` and `LoadData` procedures handle the saving and retrieving of global program data (note the use of the `TStream's Read` and `Write` methods). More robust error detection could easily be implemented by turning `SaveData` and `LoadData` into `Boolean` functions which return the value of the stream's `Status` field.

Conclusion

This article has run a bit longer than I had intended, but I wanted to cover some of the basics of streams. Armed with this knowledge, you might be well advised to go back and read the stream information contained in the Turbo Vision guide.

Hopefully, we've brought the stream concept into the light of day, and you've gained one more tool to slip into your programmer's toolbox. Use it well, and now that you've discovered streams, *wade right in.*

Don Taylor *is a consultant, author, and Pascal programmer. When he's not daydreaming in front of his terminal, you will probably find him asleep in his recliner, with a cat on his lap. He can be reached on CompuServe at 70441,1340.*

Please don't tell my mother I'm a computer programmer.

She thinks I'm a piano player in a brothel.

C/C++

The C Scape

Tim Gentry

In any craftsman's toolbox are a few prized tools. Some might be prized because they are enjoyable to use, others because they do a specific job well. Still others are there because they're the only thing available, and while they may not be needed often they are the only tools that will do the job. Some might be in the box because the craftsman made them himself, and some might be there only because they haven't gathered enough dust to be thrown away yet.

Programmer's toolkits are no exception. This month I did a bit of cleaning on the drive (it's amazing how fast 200 megs will fill...) and found that, like the toolbox above, I've got some that fit that last description. I got rid of those, but I found several other tool packages that will stick around until somebody ushers the cows into the barn.

I've done a little programming under OS/2, and while I really didn't enjoy it much I *loved* its multi-threading capability. This capability is also provided in Windows NT, some UNIX implementations, and other systems. For those of you who don't know about multi-threading here's a *very* brief description. Under multi-*tasking* operating systems many programs can be running simultaneously, each with its own isolated data space. Under multi-*threading* operating systems each executing program can have many different sub-programs (threads) running at the same time. One thread might be watching for user input, another might be watching for activity on the serial port, while yet another might be controlling output to the printer. Each thread executes independently of the others, sharing a common data space but each with its own independent stack. Usually some means of inter-thread communication is provided.

When I came back to DOS I missed that capability so much that I went out and searched for a library that even came close to providing it. I found it - in spades. Some of you might remember my discussion of CTask. Well, I'm going to recap it again. CTask is a fully preemptive multithreading system for DOS. There are no requirements for DOS extenders, no outlandish new coding styles, and (perhaps best of all) no impossibly steep learning curves. CTask takes care of coordinating all DOS calls (so that no two threads are in DOS at the same time), and provides a very extensive inter-thread communication library. Full printer and serial port functionality is available, and the package comes with full source code. With all the coding I've done with CTask I've found it extremely reliable, and absolutely indispensable. It's price? Free. Libraries just don't get any better than this.

When I started programming in Windows I found that multi-threading was again not provided but that, if anything, it was even more desirable. Back out to the networks I ran, and I found a package that starts to fill the bill. WinThreads is a non-preemptive multithreading library in the form of a Windows DLL. It has numerous disadvantages over CTask, but it is a far cry better than nothing at all. WinThreads requires you, the programmer, to make certain concessions in your coding style to work correctly. This is because Windows, while ostensibly a multi-tasking operating system, is actually a non-preemptive system. Any multi-tasking (and multi-threading, for that matter) must involve the cooperation of all of the tasks involved in order to work. If any task fails to relinquish its grasp on the CPU, all other tasks will be held from execution. This makes designing a multi-threaded system challenging, to say the least. WinThreads is a package that uses (as a start) the PeekMessage loop mechanism as the foundation for its multi-threading. The package is nowhere near as extensive as CTask, lacking almost all of its inter-thread communication functions, but for multi-threading under Windows it's the only (reasonably-priced) tool I've found that will do the job. My current project uses WinThreads to manage background modem communications at 14.4 Kbaud, and it has been quite reliable. I can only wish for better inter-thread communications.

Windows debugging can be an interesting pain, and I've got a couple of tools that help that situation. Since Windows memory management is a regular minefield of problems, I've tried to find decent tools to help out. I wished for some time that Nu-Mega would release a Windows version of Bounds-Checker, and before they finally did I found a tool that helped on that front. It's called NewTrack, and it is an extension to C++'s own memory management routines. It validates deletes, detects memory over- and under-runs, and shows up memory leaks. With all the hoops that programmers must jump through for Windows, this is a great help. Make no mistake, it doesn't have the functionality of Bounds-Checker, but it became a quick favorite of mine when in 15 minutes it helped solve a problem I'd been trying to debug for 2 weeks. Full source is provided, and the system works by adding a module to your Windows program and linking with another load library (the DLL is compiled from the provided source code).

Also on the Windows front is the Windows Debugging Kernel (or "WDK") from Microsoft. If you're *serious* about writing Windows applications and don't have this product you deserve all the problems that *will* come your way. The WDK helps prevent:

- Passing invalid parameters,
- Accessing nonexistent window words,
- Using handles after they've been deleted or destroyed,
- Using a device context after it has been released,

- Deleting GDI objects before they are selected out of a device context,
- Neglecting to delete GDI or USER objects,
- Writing past the end of an allocated memory block,
- Reading or writing using a memory pointer after it has been freed,
- Neglecting to export window procedures and other callback functions, and
- Neglecting to use MakeProcInstance with dialog procedures and other callback functions.

The WDK does have its drawbacks, including minimal (that's the nice word) documentation and the requirement for serial terminal (or second PC) to connect to the serial port for full functionality. This is an optional requirement, but in a way that's like saying that your left hand is optional equipment. You can do without it, but it makes life much easier. The other drawback is that when the WDK is active the machine is *noticeably* slower, especially in display access. But for all that it does these are small prices to pay.

The last tool that has a permanent home on my drive is the GNU RCS package. Enough has been written about it recently so I won't belabor the point, but I just can't say enough about having a good revision control system (and *using* it).

In case you're wondering where to get these tools here's a list:

CTask is public domain software by Thomas Wagner. Its main support is on BIX, but you can also get the package from CompuSpend, various Internet sites (try *oak.oakland.edu* in /pub/msdos/c), or on TUG/Pro library disk MI-CCC-DOS-073-I5.

WinThreads is shareware from Eschalon Development, Inc., 2 Renaissance Square, Suite #110, New Westminster, BC, V3M 6K3 Canada. Their phone number is (604) 520-1543. The shareware registration fee is $95.00 (US). The shareware version is available from CompuServe or on TUG/Pro library disk PT-CPP-WIN-014-I5.

NewTrack is freeware from Cavendish Software Ltd, 50 Avenue Road, Trowbridge, Wilts, BA14 0AQ, England. The software is available on the Internet (check Archie - I don't remember where I got my copy), CompuServe, and on TUG/Pro library disk PT-CPP-WIN-014-I5.

The WDK is commercial software, only available from Microsoft. It is priced right around $100.

GNU RCS (version 5.6) is freeware (subject to certain restrictions - see the file COPYING that comes with the package) by Stu Phillips. It is available from several Internet sites (get the gnuish DOS version for PCs, not the full distribution version), or on TUG/Pro library disk UT-MIS-DOS-011-I3.

***Tim Gentry** is a professional C++ programmer for Boeing Computer Services. When he isn't programming you'll probably find him at his computer working on documentation. When he isn't doing documentation he's most likely at his computer working on these columns. When he's not doing that he'll be at his computer asleep at the keyboard. He really needs to get a life.*

Library Notes

The first disk added to the library this month is disk number PT-CPP-WIN-014-I5. It contains the two tools mentioned above that aren't already on TUG/Pro library disks, namely WinThreads and NewTrack. See their descriptions above for more details. I know you'll get some use out of these two - I certainly have.

The second disk is for you Turbo Vision fans. It contains a list of known patches, bug-fixes, and enhancements to the C++ version of Turbo Vision and, unfortunately, this list is fairly large. Some of these fix memory leaks, others fix very strange behavior, and they run the entire gamut of the Turbo Vision source files. If you've been having strange results with TV lately (not Television - strange results are the norm there and I don't think we can do much about it...), you need to take a look at this list. It also contains TVRW, the Turbo Vision Resource Workshop. This is a shareware package that acts much like the dialog designer in the Windows Resource Workshop. I've also included DGLDSN, the Turbo Vision Dialog Designer. I've heard from some that TVRW is better, but I personally like DLGDSN - you get to make your own choice. Both allow you to create your dialogs on-screen, saving them for later update, and to generate source code based on the dialogs you create. Both are almost essential for TV development. The disk number for this disk is TV-CPP-DOS-003-I5.

The last disk going in this month contains a TV package that just wouldn't fit on the last disk, and a couple of Windows tools. The TV package is TVTOOLS, a collection of Turbo Vision functions that implement (among other things) password entry, intra-application communications, extended keyboard functions, fast file copy and deletion, and routines to disallow debugging of your program. I've added to this MDIDLG, a set of routines for OWL that turn dialog box definitions (created in the resource workshop) into fully MDI-compatible windows. I've been using this in one of my projects, and I can't tell you how much time this has saved. I highly recommend this class library for your C++ projects. The third package on this disk is QDHELP, a self-named "Quick and Dirty" helpfile maker. I'll also put a recommendation in for this one: while not as extensive a product as Help Magician, this one makes creating RTF files for the help compiler *much* easier. The last package on this disk is a debugging aid called DBPUTS. This DLL (with source) is handy for directing debug messages from multiple applications to a single listing, printed in the order in which they are received. This aids in (for example) debugging DDE clients and servers. This disk is TUG/Pro number TV-CPP-MIS-004-I5.

Covering the Bases

Dave Chowning

The Culture Clash Part 3

Dave Chowning
TUG/Pro Database Editor

Last year in Covering the Bases I introduced a series called The Culture Clash. In Part 1 I introduced the subject and the driving force behind it: controlling data and the desktop. I also discussed four key technologies that shape the Culture Clash: Windows, OOP, CASE and SQL. I also touched on results of technological change in the software industry. In Part 2, I discussed the response to technological change in terms of revolution versus coup. I showed how you can determine if a business is responding with a coup or a revolution. I predicted that if business and IS departments did not respond properly, the massive layoffs that we saw in 1992 would continue into 1993 and beyond, and many major businesses would fail.

I not only wrote about this, but also spoke at colleges and other places as well. I publicly laid out very clearly the steps that would occur if a company were to be saved from failure.

- Business profits would decline to the point that business failure was not only possible but inevitable if drastic action were not taken.
- The CEO would be removed and an outsider hired who understood the stockholders' fears and the need for structuring in the information age. An outsider would not have to worry about internal corporate loyalties, friendships and procedures. In fact, the new outsider CEO might bring key upper and middle level people with him.
- The new CEO would immediately begin a restructuring of the business that affected not only the direction of the business but also its scope. Key to the restructuring would be the ability to create and market new products. The speed of restructuring would be amazing until one realized that major stockholders had already been discussing its definition and implementation. More selective rather than massive layoffs would accompany the restructuring leading many uninformed analysts to ask: what's the difference? The new guy is simply cutting costs.
- New information systems policies would be designed over hardware, software, and personnel, with emphasis on downsized, distributed information systems. There would be more layoffs among systems personnel over an extended period as the new distributed systems began to replace traditional mainframe systems.
- New hiring of personnel from the outside within key areas involving new products creation and marketing and distributed information systems.

Anyone who has watched the fortunes or misfortunes of IBM during the last few years has seen my prediction come true. IBM continued massive layoffs with massive financial losses. Divisions were combined or eliminated. More layoffs. People began to talk about the end of Big Blue.

Then at the end of 1992, more layoffs were promised for 1993. IBM replaced Chairman John F. Akers with outsider Louis V. Gerstner Jr. Gerstner brought with him from RJR Nabisco his longtime media adviser, David Kalis. Analysts expect Gerstner to hire more outsiders as lieutenants. The external hiring began under ex-chairman John F. Akers, who altered a belief that outsiders rarely fit in at IBM. Akers asked business unit leaders to recruit at least two executives who someday could lead the operations. IBM has hired about 30 upper-level executives from outside the company in the last two years and has more than 50 executive searches under way.

IBM reorganized again in early 1993 and continued with more selective layoffs. With this reorganization, IBM has shifted the emphasis from mainframe computing to client server and desktop computing, even creating a Client Server unit to create and market new products. Most of IBM's 12 business units have hired outsiders, including the Consulting Group, PC Storage Devices, Adstar, Personal Software Products, Technology Products, and the Integrated Systems Solution Group.

How does this manifestation of my prediction affect IS staffing throughout the economy? At first, it will affect it drastically, as new types of professionals with new desktop and client server skills will be needed to implement new technology (see Culture Clash Part 2 for the new professionals). Later on, different types of staff will be needed to maintain and administer new systems. This may protect or prolong the jobs of some current out-of-date IS staff who can be trained over the next few years. There is developing a clear need for two distinct types of programmers: systems programmers (read that C++ Windows and Unix) and application developers (read that dBase, Paradox, FoxPro and Access, and Visual Basic). The main personnel requirements will be for people who can DO, not people who have READ, or have been PAID for it, or have STUDIED. In the end a lot of IS staff must face the fact of permanent unemployment, heavy retraining and reeducation to compete for new jobs, or new careers in lower paying service industries: "Fries with your burger, sir?" At this point we are no longer talking culture clash, but culture shock.

The biggest need of Windows programmers today is for TOOLS! Anyone trying to develop Windows applications

knows that the Resource Workshops and CodeGens of this world are not the most productive tools. Both systems programmers and application developers need powerful CASE tools for C++, data languages, SQL, and documentation. Surprisingly, the very force driving this clash is distributed information systems, i.e., distributed databases, and Windows database CASE and development tools are lagging behind Windows spreadsheets and word processors. Despite the introduction of Microsoft Access and its value as a Client Server front-end, the large mass of dBase desktop and network applications is not moving to Windows.

Where is this all leading? Armageddon? Surely, but there are some intermediate destinations: retraining, new jobs, despair, hopelessness, nursing homes, or political reeducation camps for mouthy writers.

Remember the 4 key technologies to your future: Windows, CASE, SQL and OOP. Become proficient in these areas and you will have a future. The information age is upon us. IS professionals will be needed everywhere. Even older programmers with skills will be able to work (or forced to work) past retirement age due to a shrinking population.

***Dave Chowning** is a professional database programmer and instructor who resides in Vancouver, British Columbia, Canada. He can be reached at Box 101 Station A, Vancouver, BC Canada V6C 2I8, by phone at 604/736-3200, or on CompuServe at 71672,3501.*

Product Insight: Topaz 4.0

At a Glance...

Product: Topaz
Version: 4.0

Vendor: Software Science, Inc.
1818 Gilbreth Road, Suite 230
Burlingame CA USA 94010-1213

Price: $ 199 [Topaz for Pascal]
$ 89 [Topaz for Pascal upgrade]
$ 299 [Topaz for C]

Orders: *(415) 697-0411 (orders, customer service)*
FAX: *(415) 697-1916*
CIS: 70401,1147

Software Science's Topaz is certainly the most popular and widely used of the dBASE libraries for Turbo Pascal programmers with its innovative `Tag` and `Pick` routines along with the standard full screen editing and browse capabilities. Now they have released Topaz 4.0 with two features that many of us have been waiting for: EVENT and MOUSE. Starting with this release, Topaz comes with source code for single user, multi user, and Windows engine all in the same package. As with all software, Topaz has grown so that it requires 4.5 megabytes on your hard disk.

New Units

There are new units: PRINT.TPU has been renamed to TZPRINT.TPU so that the new PRINTCOM.TPU provides an interface to the DOS PRINT.COM print spooler. Now you can add or cancel background print jobs from your programs or write utilities to do it from DOS. The low level functions in DBF4.TPU have been moved to the new TZDBFLOW.TPU. There is also a new EXTEND.TPU. Scott Bussinger has revised his EXTEND.PAS unit which allows you to open up to 250 files. It now supports DPMI.

New Features

Although there are a lot of minor new features in this release, I'll list a few of the major changes and new features starting with Mouse support, which is automatic. If a mouse driver is detected, Topaz will provide it. There are a number of mouse and event functions in the new MOUSE.TPU that allow you to work with pointing devices and disable the mouse even if the user has a mouse driver loaded.

You can use the mouse to move between `SayGet()` fields, move the cursor in fields, move around in `Browse()`, `EditText()`, and `EditMemo()`, etc. Dialog boxes support the mouse if you use the `BUTTONS` clause to specify choices. `Pick` and `Tag` as well as `PickFile` and `TagFiles` also support the mouse when it is available. You can also create mouse targets on the screen, and use the `GetEvent` function test for mouse action as well as standard keyboard action.

The BROWSE4 introduces some changes: The `Browse` function has a new look, a menu interface, and uses the ESC key to exit the FIND and SEARCH prompts. Also, `Blankfield` can be used in both prompts. The `Search` prompt is forced uppercase and the `Search` field now saves and will have the same contents the second time you choose SEARCH from the browse menu. The SEARCH now always starts at record #1. There is a new CONTINUE which calls `Search()` with `StartAtTop = false` and will continue the prior search. The `BrowseCol` function has been added to return the current browse column number. The `BrowseMenuKey` function sets the menu key, normally F10.

You can now print two reports on one page by calling `ReportForm` twice, where there is no page eject in the first report. The report will start in the middle of the page if you use a new CONNECT clause in the call to `ReportForm`. This will come in handy for those short summary style reports!

`CopyToMerge` now supports more word processor files such as MSWORD, MSWorks, Excel, and AmiPro.

Databases, memo files, and index files may now be opened read-only by setting a new global byte: `FileModeOverride := 0`.

For data entry coding with the SAYGET4 Unit, there are Fuzzy Dates which will automatically add the century to two digit years or the current year if you leave the year off. There is also a `Password` data type that results in a visible entry field showing only '*' for each character entered.

Date format support has been added for Russian, and Quicken style keystrokes have been added to the pop up

calendar (`SelectDate` function). This calendar is a great feature for data entry into `Date` fields.

Documentation

Since the Topaz package now contains three formerly separate versions (single, multi and Windows) the manual is being integrated into one 700 page, lay-flat manual. However, the documentation I received for the review was in two separate bound manuals: 184-page Users Guide, and 503- page Technical Reference. My package also has the disk file Addendum.Doc which lists many of the new functions since the last release. The README file contains installation instructions as well as some last-minute updates to the Technical Reference manual. The new Users Guide shows off the good introduction to using Topaz that has previously been at the front of the reference manual. New users to Topaz should not be intimidated by the large number of features in this library/document. Both bound manuals have good tables of contents and indexes which make using them quite easy. The functional listing by type of routine (printing, data entry, report etc.) has been included at the front of the Technical Reference manual. Only with the release of separate bound manuals do you begin to appreciate some of the work that has gone into the documentation of Topaz over the years.

The biggest drawback to the Topaz documentation is the lack of screen shots and illustrations. Although the style remains consistent from version to version, making it easy to use the manuals, they resemble word processing documents and not manuals. I have recently been using tutorial books on various Windows software which are fantastic in the use of illustrations. This new platform (Windows) forces us to not only reevaluate our software tools, but also the documentation and help features of those tools.

On the Balance

Topaz is a fantastic routines library for not only database programming, but also for interface routines such as picklists, taglists, strings, calculator, calendar, print spooler, and text manipulation. The two utilities CREATE (dBase data files) and MAKEPAS code generator are helpful tools in getting those quick and dirty applications up and running. Topaz 4.0 gives you added control over the templates used for generating: UNIT, INCLUDE, or PROGRAM code files. You don't need to be a dBASE programmer to use Topaz successfully the first shot out of the box!

Despite the new features and documentation, there is no significant Windows interface, the Topaz Report Generator has not been significantly updated for several years, and there is still no support for standard dBASE/Clipper/FOX index files. I again found errors in compiling a couple example programs, and this is the second straight release where I have been unable to successfully compile the help utility.

Software Science president George Rothbart told me, "You are very correct about the report generator. We put all our effort into good mouse support (automatic everywhere, but with a nice interface so that you can easily add your own mouse targets and mouse handlers), and in supporting DPMI, etc. I have requests to boost the functionality of the report generator, and I believe it is time to take them seriously."

Future Releases

As for the future of Topaz, George said, "We have been working practically around the clock since June of 1992 on Topaz for C. That's right, C. And we've got it! We took the time to optimize indexing and packing, and the C version gives you a factor 5 speed improvement in these activities (and are now faster than CodeBase and similar to FoxPro). Topaz for C comes with its own 800 page manual, full source code, pre-compiled libraries for the Borland and Microsoft C or C++ compilers, examples, utilities, the works. We will be releasing the C product on May 16th, the first day of the Borland conference in San Diego."

"After we come back from San Diego, we will plan out our R&D activities for the rest of 1993. These include:

- Support for NDX index files (positively the very next feature) This will happen first for the C version, and then be ported to the Pascal version.
- Better EMS/XMS support in Topaz (take advantage of EMS/XMS for data buffers, index caches, etc. We hope to be able to have 128 databases open simultaneously).
- Migrate to object orientation. OOPize Topaz functions. Better Windows support. Currently only the database engine works in Windows ... we want to improve on this.
- User fonts, and a fat-bit editor.
- A RUN command (like TurboPower's `ExecDosSwap`)"

A Gold Mine of Routines

I have used the print spooler routines, the editor and the page image routines extensively in my code generating and documentation utilities. I have found no match for those particular routines in any other Pascal library. Keep the Topaz manuals next to your keyboard. They contain a gold mine of routines that will come in handy at any time. ***- DC***